VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Spring Framework a vývoj webových aplikací

Spring Framework and Development of Web-Based Applications

2011 Lukáš Zapletal

„Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.“

Abstrakt

Má bakalářská práce se zabývá vývojem webových aplikací ve Spring frameworku a jedním z jejich cílů je vytvoření informačního systému pro zaznamenávání a publikování sportovních výsledků, novinek ze světa sportu, týmových sestav, soupisek hráčů apod. Práce také pojednává o historii, vývoji, specifikacích a modulech tohoto frameworku. V další části je prováděno srovnání Spring MVC frameworku s jinými MVC frameworky pro tvorbu webových aplikací. V poslední části se práce zabývá popisem specifikací a vytvořením analýzy webové aplikace.

Klíčová slova

Spring framework, webová aplikace, Inversion Of Control

Abstract

My bachelor thesis deals with developing web applications in the Spring framework and one of its objectives is the creation of an information system for recording and publishing sports results, news from the world of sport, team reports, player squads, etc. The work also discusses the history, development, specifications and building blocks this framework. The next part is made of Spring MVC framework compared to other MVC frameworks for building Web applications. In the last part of the work deals with the description of specifications and analysis of creating a web application.

Key words

Spring framework, Web-Based Appliaction, Inversion Of Control

Obsah

1 Úvod

Sport je v současné době velmi oblíbeným způsobem trávení volného času, avšak pokud se budeme věnovat jen jeho pasivní složce, to je například rozboru různých výsledků utkání, sestavám hráčských týmů, pravidlům jednotlivých disciplín apod., bez využití výpočetní techniky a moderních informačních technologií se ani zde neobejdeme. A právě této rovině se věnuje má bakalářská práce, která si dala za úkol vytvořit webovou aplikaci sloužící tyto informace zajistit.

Celá práce je rozdělena na úvod, 4 kapitoly a závěr:

V první kapitole jsem popsal historii a vývoj Spring frameworku a dále jsem zde vysvětlil jeho moduly a specifika.

Druhá kapitola nabízí srovnání Spring MVC frameworku s ostatními MVC frameworky. Pro toto srovnání jsem si vybral čtyři MVC frameworky, které využívají jazyk Java při tvorbě aplikací – jsou to Struts 2, Jboss Seam, Apache Wicket a Apache Tapestry.

Třetí kapitola se zabývá specifikacemi požadavků na mnou vytvořenou webovou aplikaci, která slouží k zaznamenávání sportovních výsledků, jejich publikování apod. Snažil jsem se popsat webovou aplikaci pro uživatele přehlednou, snadno použitelnou a hlavně účelnou.

Čtvrtá kapitola se zabývá analýzou a návrhem řešení webové aplikace, součástí této kapitoly je i datová a funkční analýza. Datová analýza se zabývá databázovou strukturou, která je popsána třídními diagramy dále obsahuje lineární zápis entit a také datový slovník. Funkční analýza popisuje pomocí sekvenčních diagramů konkrétní případy užití.

2 Specifika Spring Frameworku a jeho využití při vývoji webových aplikací

2.1 Úvod do Spring Frameworku

Spring je aplikační rámec (framework) pro vývoj aplikací nad platformou Java. Je určen zejména pro vývoj J2EE (Java 2 Enterprise Edition) aplikací. Byl vyvinut především pro usnadnění vývoje a k lepší orientaci v procesu tvorby enterprise aplikací. Framework má „otevřený zdrojový kód“ (open-source) a patří mezi tzv. odlehčené J2EE kontejnery (lightweight). Na jedné straně by se mohlo zdát, že užití „Springu“ pozbývá významu při vývoji webových aplikací, avšak framework nabízí mnoho možností např. deklarativní správu transakcí, využití webových služeb, poštovních služeb, vzdálený přístup k aplikační logice s využitím RMI (Remote Method Invocation) a zároveň i hodně variant pro přístup k perzistenci dat v databázi. Při komunikaci aplikace s prezentační vrstvou, Spring nabízí plnohodnotný MVC framework (model – view- controller), velmi dobrou strukturu výjimek a dále také mnoho možností integrace AOP (Aspect-Oriented Programming) přístupů, které se využívají zejména při implementaci informačních systémů apod.

Spring můžeme charakterizovat jako modulární systém složený z několika základních modulů – blíže viz kapitola 1.4. Díky jeho modulárnosti můžeme použít při vytváření informačních systémů například jen některý, aniž by byla nutná další funkční závislost na jiné moduly. Tato vlastnost „Springu“ nabízí programátorovi úplnou volnost a proměnlivost u návrhu a vývoje informačních systémů. Spring umožňuje dále i spolupráci s technologiemi různých vrstev. Zjednodušeně bychom mohli říci, že Spring je zprostředkovatelem mezi aplikacemi a technologiemi - např. Hibernate, JSF (Java Server Faces), EJB (Enterprise Java Bean).

Spring používá celá řada velkých společností pro vývoj svých enterprise aplikací. Jedná se například o velké světové bankovní instituce, významné pojišťovny v Evropě i v Americe. Dále ho používají i některé významné úřady, jako například European Patent Office, French Online Taxation Systém, americká, kanadská a australská vládní agentura, společnosti Oracle, eBay, CERN (European Organization for Nuclear Research) apod. Spring také vyvíjí několik zajímavých projektů, Spring Web Flow - framework pro management toku webové aplikace, Spring.NET – Spring framework určený pro architekturu .NET, z dalších jsou nejznámější Spring Rich Client, Spring IDE for Eclipse a Spring OSGi. O dalších projektech je možno zjistit více na oficiálních webových stránkách http://www.springsource.org.

2.2 Historie vývoje Spring Frameworku

Tento framework v jeho první verzi popsal v říjnu 2002 známý programátor Rod Johnson ve své publikaci Expert One-on-One J2EE Design and Development[odkaz]. V této knize se autor zabývá především vývojem J2EE aplikací. Popisuje zde běžné problémy programátorů, se kterými se setkávají při tvorbě J2EE aplikací. Dále je zde popisován kód rámce, který má název Interface21. Tento rámec by měl usnadnit programátorovi vývoj J2EE

aplikací. Později byl tento framework zdokonalen a rozšířen a dostal název Spring Framework a je šířen jako open-source.

Spring byl šířen v roce 2003 pod licencí Apache 2.0, v březnu 2004 byl vydán ve verzi 1.0 a v září 2004 a březnu 2005 byly uvolněny jeho další verze. Verze Spring Frameworku 1.2.6 získala ocenění Jolt Productivity Awards a také JAX Innovation Award. V současné době je nejaktuálnější verze 3.0.x - přesněji 3.0.5.RELEASE.[1]

Tabulka 1 Vývoj jednotlivých verzí:

Rok	Verze	Popis
2002	Interface21	1. Verze – Rod Johnson
2004	1.0	Licence Apache 2.0
2004,2005	Další verze	Přístupné pro vývojáře
2006	1.2.6	Získala dvě ocenění
2008	2.0.x	7. 1. 2008 poslední verze 2.0.8 kompatibilní s Java 1.3 a vyšší
2008	2.5.x	31. 10. 2008 poslední verze 2.5.6 kompatibilní s Java 1.4 a vyšší
2009	3.0.x	20. 10. 2010 poslední verze 3.0.5 vyžaduje Java 1.5 a vyšší

2.3 Důvody vzniku Spring Frameworku

Framework vznikl hlavně pro usnadnění vývoje a lepší orientaci při tvorbě enterprise aplikací.

Základní usnadnění vývoje za použití Spring Frameworku:

- Využití návrhového vzoru Inversion of Control (blíže viz kapitola 1.4), který pomáhá odstranit těsné programové vazby POJO (Plain Old Java Objects – blíže viz kapitola 1.5) objektů a vrstev
- Architektura nepředepisuje implementaci – můžeme si vybrat způsob implementace (EJB, POJO) business vrstvy pro architekturu aplikace
- Nemusíme používat EJB pokud řešíme různé aplikační domény – transakční zpracování, podpora RMI, webových služeb
- Pro přístup k datům je možná implementace vlastních komponent. Přístup k datům může být řešen přímo pomocí JDBC (Java Database Connectivity) nebo pomocí ORM (Object – Relation mapping) technologií nebo nástrojů, např. Hibernate, TopLink, JDO
- Je zrušena závislost na nepřehledných konfiguracích, složité vyhledávání významu každé z nich, což je velká úspora času pro programátora
- Abstrakce vede ke snadnějšímu využití částí J2EE – např. JMS (Java Messaging Services), JMX (Java Management Extensions), JavaMail, JDBC
- Snadnější kódování i používání JUnit testů
- Business komponenty – možnost jejich správy a konfigurace [2] [3]

[1] Url: http://www.theserverside.com/news/1364527/Introduction-to-the-Spring-Framework. [cit. 1. 3. 2011].

[2] Url: http://www.cs.vsb.cz/behalek/frvs/2005/java/spring/spring.html. [cit. 1. 3. 2011].

[3] Url: http://interval.cz/clanek.asp?article=4045. [cit. 1. 3. 2011].

2.4 Návrhový vzor Inversion of Control

Inversion of Control, často se říká IoC kontejner, je návrhový vzor, který využívá jádro Spring Frameworku. Ve volném překladu by Inversion of Control znamenalo obrácené řízení.

Hlavním principem Inversion of Control je přesunout zodpovědnost za vytváření, inicializaci a propojení objektů z aplikace na framework. Zjednodušeně můžeme říci, že řeší těsné vazby mezi objekty aplikace, které jsou ve zdrojovém kódu těsně svázány. Jednotlivé objekty je tak možno získat pomocí tzv. svazování závislostí.

Spring framework používá pro konfiguraci IoC kontejneru metadata. Metadata mohou být např. XML, anotace, properties. IoC má své objekty, tzv. managované objekty, které se ve Springu nazývají beany. IoC kontejner se startuje programově, to znamená, že má připravená různá API, např. WEB – Servlet nebo JUnit – Speciální TestCase. Také jej lze spouštět přímo.

Příklad přímého spuštění:

```
XmlBeanFactory bf = new XmlBeanFactory("applicationContext.xml");
MyBusinessObject mbo = (MyBusinessObject) bf.getBean("exampleBusinessObject");
```

Pokud nepoužíváme návrhový vzor IoC a zůstaneme u klasického způsobu programování, musíme nejprve vytvořit třídu, která poté využívá další třídy, a ty využívají zase další a další. Tento postup pak vytváří velmi pevné vazby a změna jedné třídy za jinou vyžaduje úpravu celého kódu. IoC právě tyto vazby umožnuje uvolnit. V tomto případě se používá slovní spojení „Hollywoodský přístup – Nevolejte nám, my se ozveme“. Třída totiž sama nevytváří instanci dalších tříd, které potřebuje, ty jsou jí dodány jedním ze tří nejznámějších způsobů z vnějšku – Constructor Injection, Setter Injection, Interface Injection.

2.4.1 Constructor Injection

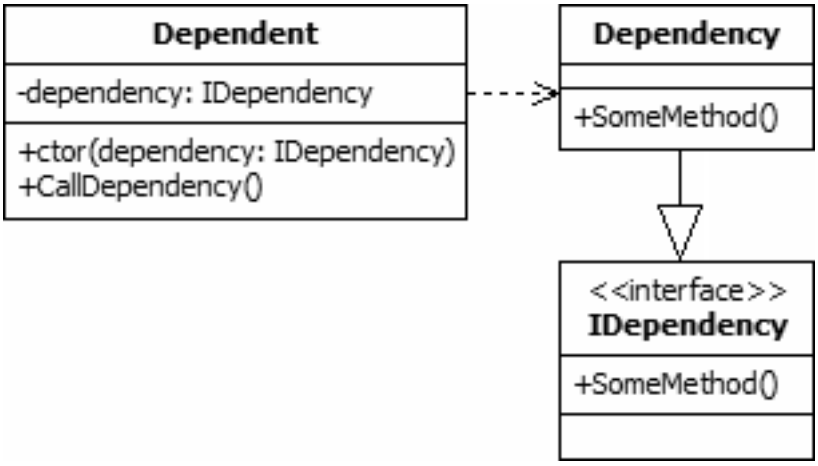
Constructor Injection je způsob, kdy každá třída, do které je vložena instance nějaké třídy, musí mít předem vytvořený konstruktor, který umí pracovat s konkrétními typy objektů. Tomuto konstruktoru musíme nastavit parametry, předtím, než budeme vytvářet instanci objektu. Třída, která potřebuje „závislost“ (Dependency) musí mít veřejný konstruktor přebírající instanci závislosti (Dependency) jako argument tohoto konstruktoru (viz Obrázek 1). Toto využívá například PicoContainer. Spring samozřejmě tento způsob podporuje taky.

Třída Dependent – tato třída reprezentuje závislé třídy, závislost je aplikována pomocí konstruktoru

Rozhraní IDependency – toto rozhraní definuje závislé metody

Třída Dependency – tato třída představuje závislost, která může být aplikována na závislý objekt

Obrázek 1 - Obecný příklad Constructor Injection



Zdroj: Url: http://www.blackwasp.co.uk/DependencyInjection.aspx.

2.4.2 Setter Injection

Setter Injection je nejčastěji používaný případ Dependency Injection ve Spring frameworku. Způsob vkládání objektů pomocí Setter Injection spočívá v tom, že každá třída, do které jsou vkládány instance jiných tříd, musí mít definovány metody set (). Pomocí těchto metod je poté prováděno vkládání instancí (viz Obrázek 2).

Hlavní rozdíly oproti Constructor Injection

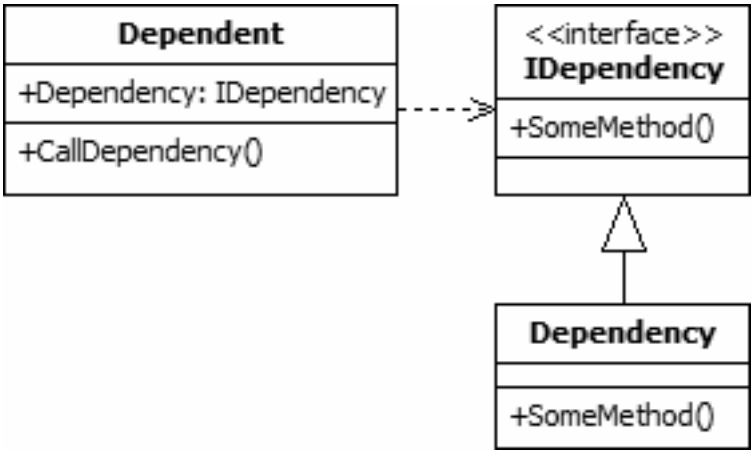
- Metody set () mohou být děděny, zatímco konstruktory nikoliv
- Konstruktory s mnoha parametry, zejména stejného typu, mohou být nepřehledné nebo i špatně používané
- Při použití Setter Injection může být závislý objekt změněn za běhu

Třída Dependent – tato třída reprezentuje závislé třídy, závislost je aplikována pomocí vlastnosti Dependency

Rozhraní IDependency – toto rozhraní definuje závislé metody

Třída Dependency - tato třída představuje závislost, která může být aplikována na závislý objekt

Obrázek 2 - Obecný příklad Setter Injection



Zdroj: Url: http://www.blackwasp.co.uk/DependencyInjection.aspx.

2.4.3 Interface Injection

Použití Interface Injection spočívá v tom, že nejprve musíme vytvořit rozhraní, ve kterém jsou definovány metody pro nastavení instancí závislých tříd. Třída, která potřebuje pracovat s těmito závislými třídami, musí implementovat vytvořené rozhraní. Tento případ vkládání závislostí používá nejčastěji Avalon framework. Spring framework Interface Injection nepodporuje.

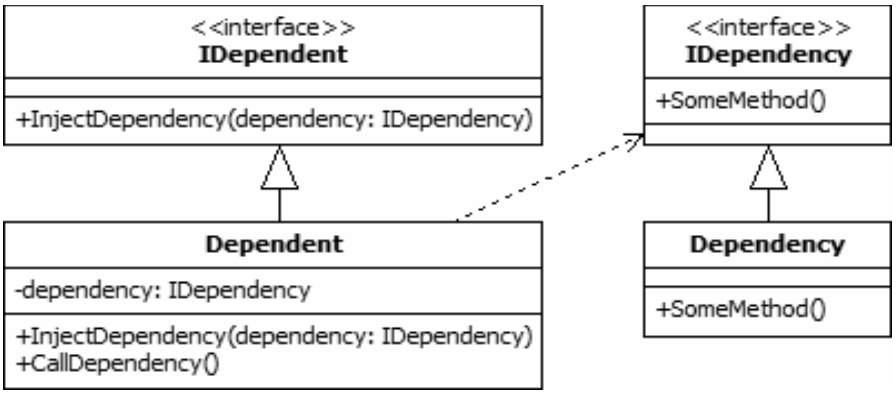
Rozhraní IDependent – toto rozhraní definuje metodu, která se používá ke vkládání jedné nebo více závislostí

Třída Dependent – tato závislá třída implementuje rozhraní IDependency

Rozhraní IDependency – toto rozhraní definuje závislé metody, pomocí rozhraní můžeme snadno měnit typy závislostí

Třída Dependency - tato třída představuje závislost, která může být aplikována na závislý objekt

Obrázek 3 - Obecný příklad Interface Injection



Zdroj: Url: http://www.blackwasp.co.uk/DependencyInjection.aspx.

2.4.4 Dependency Injection

Dependency Injection se užívá jako novější název pro IoC, v překladu to znamená vkládání závislostí. Dále Dependency Injection zužuje okruh, na který se vztahuje, protože se jedná již o určitou techniku využití IoC. Dependency Injection se stará i o vložení nebo také spojení potřebných objektů.

Výhody Dependency Injection

- Menší závislost mezi komponentami, proto je jejich programování a údržba snadnější
- Jednodušší testování zdrojového kódu
- IoC kontejner se stará o přetypování tříd – není třeba se tímto dále již zabývat

Nevýhody Dependency Injection

- Horší orientace ve zdrojovém kódu pro programátory, kteří návrhový vzor IoC neznají
- Nevyplatí se používat pro snadější aplikace – zde je lepší využít klasického způsobu programování v jazyce Java
- Složitý vývoj aplikací ve vývojovém prostředí (IDE), které IoC nezná [4] [5] [6]

2.5 Moduly Spring Frameworku

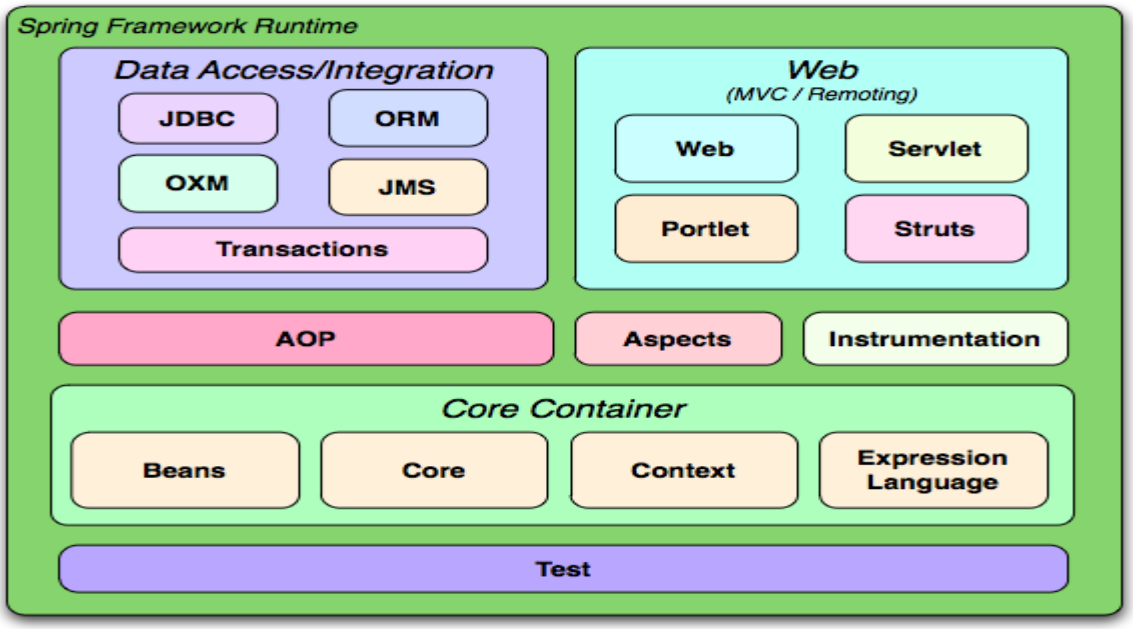
Architektrua Spring Frameworku nám nabízí mnoho různých možností a funkcionalit. Spring se skládá z přibližně 20 modulů. Tyto moduly jsou seskupeny do Core container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation a Test (Obrázek 4). Kterýkoli z modulů můžeme používat v rámci kontejneru. Inicializace modulu se provádí přidáním jednoho řádku do kódu. Život všech objektů, které Spring spravuje, se odehrává v prostředí kontejneru.

[4] Url: http://interval.cz/clanky/spring-framework-predstaveni-j2ee-lightweight-kontejneru/. [cit. 4. 3. 2011].

[5] Url: http://sqdw.signaly.cz/0804/inversion-of-control-dependency. [cit. 4. 3. 2011].

[6] Url: http://www.theserverside.com/news/1364527/Introduction-to-the-Spring-Framework. [cit. 4. 3. 2011].

Obrázek 4 - Moduly Spring Frameworku



Zdroj: Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/overview.html#overview-modules.

2.5.1 Core Container

Core Container se skládá z modulů Core, Beans, Context a Expresion Language.

Moduly Core a Beans

Tyto moduly jsou základními moduly Spring Frameworku, představují celý základ systému a umožňují vlastnost IoC a Dependency Injection (blíže viz kapitola 1.4). O svazování vazeb mezi každým bean objektem se stará právě tato vlastnost, a ta se také stará o správu a funkčnost v rámci celého Core kontejneru.

Základem tohoto modulu je implementace BeanFactory. BeanFactory je velmi propracované provedení návrhového vzoru factory (továrna). BeanFactory zajišťuje životní cyklus POJO objektů tj. vytváří objekty, nastavuje vazby mezi objekty, incializuje objekty, poskytuje objekty k použití a pokud se kontejner zastaví, objekty ukončuje.

Příklad inicializace Springu

Při vytváření webové aplikace musíme konfigurační soubor umístit do adresáře /WEB-INF. Do tohoto adresáře nemá prohlížeč přístup, což vyplývá ze specifikací J2EE.

Příklad:

```
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.web.context.support.ServletContextResource;
public void init ( ) {
        XmlBeanFactory bf = new XmlBeanFactory ( new ServletContextResource ( this.getServletContext(), "/WEB-INF/applicationContext.xml"));
}
```

Modul Context

Tento model obsluhuje kontext celého systému a zajišťuje přístup k jednotlivým bean objektům v rámci aplikace. Dal by se přirovnat k JNDI (Java Naming and Directory Interface), pokud používáme jiný systém, nežli je Spring Framework. Context modul má všechny vlastnosti modulu Beans, ale poskytuje navíc podporu internacionalizace a J2EE funkce, např. EJB, JMX. Hlavním bodem tohoto modulu je rozhraní ApplicationContext.

Modul Expresion

Tento modul poskytuje rozšíření jazyka pro vytváření výrazů (unified expresion language). Je pod specifikací JSP 2.1. Jazyk také podporuje volání metod, aritmetické a logické operátory a vyhledávání objektů ze Spring IoC kontejneru. [7] [8] [9]

2.5.2 Data Access / Integration

Data Access / Integration se skládá z modulů JDBC, ORM, OXM, JMS a Transaction.

Modul JDBC

JDBC modul nám nabízí abstraktní JDBC vrstvu, čímž odpadá složité JDBC kódování a odchytávání výjimek specifických pro každou databázi. Následující tabulka ukazuje, o co se musí starat programátor a co za programátora dělá Spring.

[7] Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/beans.html#beans-introduction. [cit. 7. 3. 2011].

[8] Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/beans.html#context-introduction. [cit. 7. 3. 2011].

[9] Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/expressions.html. [cit. 7. 3. 2011].

Tabulka 2 Spring a programátor

Akce	Spring	Programátor
Definovat parametry připojení		X
Otevřít spojení	X	
Vytvoření SQL dotazů		X
Deklarace parametrů, přiřazení hodnot		X
Příprava a spuštění dotazu	X	
Obsluha výjimek	X	
Obsluha transakcí	X	
Ukončení připojení	X	

Zdroj: http://static.springsource.org/spring/docs/3.1.0.M1/spring-framework-reference/html/jdbc.html#jdbc-introduction

Modul ORM

Tento modul umožňuje integrovat vrstvu pro objektově-relační mapování (ORM) objektů z frameworků jako je Hibernate, iBatis, Java Persitence API, Java Data Objects. Při využití těchto ORM frameworků je samozřejmě možné využít i všech specifik, které Spring Framework nabízí, jako je například podpora transakcí nebo AOP (Aspektově orientované programování).

Modul OXM

Z pomocí tohoto modulu můžeme implementovat Object/XML mapování pro JAXB, XMLBeans a Xstream. Je to vlastně převod XML dokumentu na objekt a naopak. Někdy se tento proces převodu nazývá XML Marshalling nebo XML Serializace. XML dokument může být strukturován pomocí DOMu (Document Object Model) nebo vstupního či výstupního proudu. Mezi výhody použití tohoto modulu patří snadná konfigurace, konzistentní rozhraní a konzistentní hierarchie výjimek.

Modul JMS – Java Messaging Service

Spring Framework poskytuje integraci JMS frameworku, který zjednodušuje používání JMS API. JMS neslouží k posílání emailů, přesto zde komunikace probíhá podobně jako u emailů. Komunikace mezi klienty je v podobě asynchronních zpráv, což mohou být například požadavky, události, odpovědi a jiné. Zprávy jsou vytvářeny a užívány aplikacemi, dále také obsahují informace, pomocí kterých tyto aplikace fungují, nastavují svůj stav a koordinují svou činnost v rámci jednoho celku.

Modul Transaction

Pro všechny třídy, které implementují speciální rozhraní nebo POJO objekty, poskytuje tento modul programové a deklarativní řízení transakcí. Hlavní výhodou modulu je konzistentní programovací model pro různé transakce, například v Hibernate, JPA, JTA, JDBC, podpora deklarativního řízení transakcí, jednodušší API pro toto řízení transakcí apod. [10] [11] [12] [13] [14]

2.5.3 Web

Tato vrstva Springu obsahuje moduly Web, Web – Servlet, Web – Struts a Web – Portlet.

Modul Web

Tento modul nabízí základní funkce využitelné při tvorbě webových aplikací. Poskytuje nám např. funkci Multipart file upload, incializaci IoC kontejneru pomocí „servlet listeners“ nebo práci s cookies.

Modul Web – Servlet

Web – Servlet obsahuje implementaci Model – View – Controller (MVC) pro webové aplikace. Spring MVC Framework poskytuje oddělení kódu doménového modelu a webových formulářů s možností využití všech prvků Spring Frameworku.

Modul Web – Struts

Tato část nabízí podpůrně třídy, pomocí kterých můžeme integrovat Struts Framework do webových aplikací postavených na Springu. Tato podpora je nyní zastaralá, proto je lepší využít Struts 2.0 framework a integraci do Springu nebo využít Spring MVC Framework.

Modul Web – Portlet

Portlet je webová komponenta, která se využívá jako výměnná komponenta uživatelského rozhraní, poskytující prezentační vrstvu pro informační systémy (IS). Spolupráce aplikace a portletu zajišťuje API definované Java Portlet Specifikation.

---

[10] Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/jdbc.html#jdbc-introduction. [cit. 14. 3. 2011].

[11] Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/orm.html#orm-introduction. [cit. 14. 3. 2011].

[12] Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/oxm.html. [cit. 14. 3. 2011].

[13] Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/jms.html. [cit. 14. 3. 2011].

[14] Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/transaction.html. [cit. 14. 3. 2011].

Web – Portlet nabízí implementaci model – view – controller pro prostředí portletu. Má podobné funkce jako modul Web – Struts. [15] [16]

2.5.4 AOP (Aspect-oreinted programming) a Instrumentation

Tento modul je jedna z nejsilnějších vlastností Spring Frameworku a je využíván napříč celým frameworkem. AOP poskytuje podporu pro aspektově orientované programování, což znamená, že umožňuje rozdělení částí zdrojového kódu, který je pak používán v celé aplikaci (autorizace, transakce), na jednotlivé aspekty, a ty následně využít pro jakýkoliv POJO objekt.

Test

Testování komponent Spring Frameworku pomocí JUnit nebo TestNG se provádí pomocí tohoto modulu. Test umožňuje vytvoření „falešných objektů“, které můžeme využít při testování kódu.

2.6 Webová vrstva aplikace – Spring Web MVC

V této kapitole se budu zabývat vývojem webových aplikací pomocí Springu, konkrétně jeho modulem Web MVC (model – view - controller). Tento modul nebo kterýkoliv jiný nám umožňuje, plně využívat všech možností a služeb Spring Frameworku. Popisuji zde pouze základní části webové aplikace postavené na Springu.

2.6.1 MVC – model – view - controller

Návrhový vzor MVC v klasických webových aplikacích, založených na request/respond (požadavek/odpověď), rozděluje celou aplikaci na 3 části – Model, View, Controller.

- Model – je vytvořen z objektů, které obsahují data. Tyto data budou zobrazena pomocí pohledu (View).
- View (pohled) – view přebírá od kontroleru model a poté data z modelu zobrazí (zformátuje), tj. zasílá odpověď.
- Controller (kontroler) – kontroler zpracovává požadavek, ze kterého získává údaje. Tyto údaje využívá při tvorbě modelu, který poté předává do pohledu.

2.6.2 Třída DispatcherServlet

Základní třídou Spring MVC frameworku je DispatcherServlet (*org.springframework.web.servlet.DispatcherServlet*), který musí být definován v konfiguračním souboru web.xml. Výhodou této třídy je, že může mít v rámci jedné aplikace

[15] Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#mvc-introduction. [cit. 14. 3. 2011].

[16] Url: http://interval.cz/clanek.asp?article=4045. [cit. 14. 3. 2011].

definováno několik servletů a každému z nich je pak přiřazen jeho vlastní aplikační kontext, který je nezávislý. Díky této výhodě je umožněna modularizace webového rozhraní.

Příklad:

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
// pokud jsme pojmenovali servlet jako dispatcher, musí se konf. soubor jmenovat
// dispatcher-servlet.xml
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>*.htm</url-pattern>
  </servlet-mapping>
```

Pomocí toho kódu jsme definovali servlet s názvem dispatcher, který je instancí třídy DispatcherServlet a tento servlet je namapován tak, že přijímá všechny požadavky s příponou .htm. Pomocí parametru *contextConfigLocation* jsme nastavili soubory, které definují webový aplikační kontext daného servletu. Aplikační kontext slouží k definici objektů webové vrstvy. Inicializace aplikačního kontextu probíhá při spuštění servletu.

2.6.3 Zpracovávání požadavků

Pokud Dispatcher Servlet přijme požadavek (jakýkoliv objekt typu *javax.servlet.http.HttpServletRequest*) je zpracováván takto:

Nejprve se umístí aplikační kontext servletu do atributu *DispatcherServlet.WEB_APPLICATION_CONTEXT_ATTRIBUTE*. Do tohoto atributu mají přístup i jiné objekty, které mají vliv na životní cyklus požadavku.

Objektu požadavku se přidá jako atribut „detektor národního prostředí." Je to objekt, který implementuje rozhraní *org.springframework.web.servlet.LocaleResolver*. Tento objekt musí mít definici v aplikačním kontextu s názvem *localeResolver*. Využívá se pro rozeznání národního prostředí klienta. Pokud tato definice neexistuje, je použita instance třídy *org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver*.

Další atribut, který je přidán objektu požadavku, je „detektor motivů", objekt implementující rozhraní *org.springframework.web.servlet.ThemeResolver*. Pokud nepoužíváme žádné motivy (themes) můžeme tento atribut ignorovat.

Pokud v aplikačním kontextu definujeme objekt s názvem *multipartResolver*, který implementuje rozhraní *org.springframework.web.multipart.commons.CommonsMultipartResolver* je tento objekt požadavku dále zabalen do *MultipartHttpServletReques*. Tímto je usnadněna práce se soubory, které jsou posílány na server. Pokud v aplikačním kontextu není definován objekt *multipartResolver*, požadavek není ovlivněn.

Všechny objekty, které se nacházejí v aplikačním kontextu a jsou typu *org.springframework.web.servlet.HandlerMapping* se dotazují na kontroler. Kontroler požadavek zpracuje. Dále se objekty dotazují na interceptor, který provádí předzpracování a postzpracování požadavku. Pokud není definován žádný objekt typu *HandlerMapping*, vytváří se instance třídy *org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping*.

Poté se požadavek předává interceptoru, pokud byl nějaký vybrán. Interceptor provede předzpracování, předtím, nežli je předán do kontroleru. Interceptor může provést jakoukoliv akci, například přesměrování požadavku jinam.

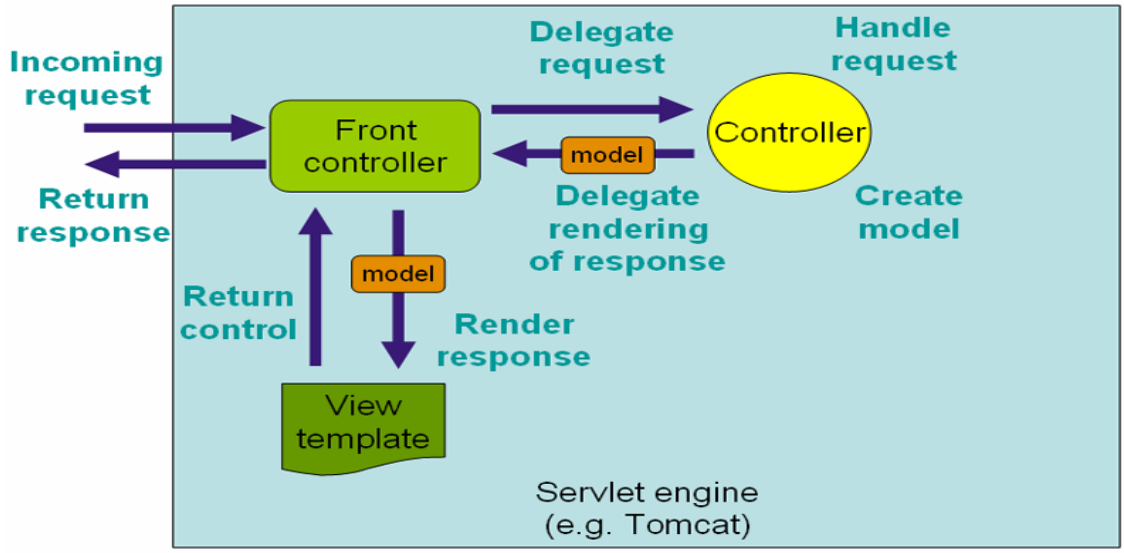
Požadavek se pak předává objektu typu *org.springframework.web.servlet.HandlerAdapter*, který se hledá v aplikačním kontextu. O určení typu kontroleru rozhoduje implementace tohoto rozhraní. Toto rozhraní nám umožňuje snadnou záměnu Spring MVC Frameworku za jiný, který má jiné typy kontrolerů. *HandlerAdapter* se vyhledává podle svého typu v aplikačním kontextu. Defaultní implementace rozhraní *je SimpleControllerHandlerAdapter*, podporující kontrolery Spring MVC, které implementuje rozhraní *Controller*.

Vybranému kontroleru je předán požadavek k dalšímu zpracování. Kontroler vytváří model spolu s aplikační vrstvou aplikace a vytvoří logický název pohledu. Tento název bude použit pro zobrazení modelu. Model a logický název je předáván dále pomocí tohoto kontroleru. Interceptor opět zpracovává požadavek, pokud byl nějaký vybrán.

Poté v aplikačním kontextu nastává opět prohledávání a jsou vyhledávány všechny „detektory pohledu“, objekty typu *ViewResolver*. Tyto objekty převádí logický název pohledu na konkrétní cestu k souboru, který provádí zobrazení. Pro zobrazení se většinou používá některá ze šablonovacích technologií, nejčastěji JSP, ale také např. FreeMarker, Velocity.

Konkrétním pohledem je tedy šablona daného jazyka. Pomocí konkrétního pohledu je zobrazen získaný model. [17] [18]

Obrázek 5 Zpracování požadavku



Zdroj: Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#mvc-servlet.

[17] Url: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#mvc-introduction. [cit. 20. 3. 2011].

[18] Url: http://morosystems.cz/java/spring/ch03s03.php. [cit. 20. 3. 2011.]

2.6.4 Kontrolery

Kontrolery jsou jedny z klíčových částí návrhového vzoru MVC (model – view - controller), proto Spring Framework nabízí komplexní hierarchii rozhraní a tříd, které nám při implementaci webové aplikace usnadňují práci**.**

Klíčovým rozhraním je *org.springframework.web.servlet.mvc.Controller*. S tímto rozhraním spolupracuje *DispatcherServlet*, přesněji *SimpleControllerHandlerAdapter*. Toto rozhraní má pouze jednu metodu *ModelAndView handleRequest(HttpServletRequest req, HttpServletResponse res)*.

Tato metoda nahrazuje metody *doGet* a *doPost* třídy *javax.servlet.http.HttpServlet*, protože je jí předáván odkaz na objekty požadavku i odpovědi.

Třída *ModelAndView* je dvousložková komponenta, která obsahuje model i pohled. Model je většinou instance třídy *java.util.Map*. Pohled mívá dvě různé formy. První forma je konkrétní objekt implementující rozhraní *org.springframework.web.servlet.View*, která se stará přímo o zobrazení modelu. Druhý způsob je symbolický název pohledu. Tento název bude zpracován až „detektorem pohledu“ (objekt typu *ViewResolver*) a převeden na konkrétní objekt typu *View*. [19]

Příklad jednoduchého kontroleru:

```
public class PozdravController implements Controller {
        public ModelAndView handleRequest (HttpServletRequest req, HttpServletResponse
res)
throws Exception {
        String pozdrav = “Ahoj fanousci fotbalu”;
        Map model = new HashMap();
        model.put(“pozdrav”, pozdrav);
        return new ModelAndView(“PozdravFotbal”, model);
}
}
```

[19] Url: http://morosystems.cz/java/spring/ch03s09.php#kontrolery. [cit. 1. 4. 2011].

3 Porovnání vlastností Spring frameworku a ostatních MVC frameworků pro vývoj webových aplikací

3.1 Dělení a výhody Model – View - Controller Frameworků

Pří dělení MVC Frameworků je rozhodující úroveň abstrakce nad protokolem HTTP, podle toho se dělí frameworky na dva specifické typy – požadavkově orientované a komponentově orientované.

3.1.1 Požadavkově orientované (Request based)

V těchto typech frameworků je příchozí požadavek (request) zpracováván přímo kontrolerem (controller). Tento způsob komunikace znamená, že framework dostane od uživatele požadavek (request) a podle něj se určí, co bude systém dělat. Poté odešle uživateli odpověď (response). Mezi request based rámce patří například Apache Struts, Stripes, Spring MVC, Grails, Jboss Seam a další.

3.1.2 Komponentově orientované (Component based)

Tyto frameworky při vývoji webové aplikace připomínají svými rysy vývoj desktoptových aplikací.

Požadavkově orientované frameworky jsou starší, nežli komponentově orientované, které se začaly využívat až v posledních letech. Typickými představiteli těchto typů rámců jsou například. JSF (Java Server Faces), Tapestry, Wicket.

3.2 Proč využívat MVC frameworky při tvorbě webových aplikací

Využití frameworku při tvorbě webové aplikace nabízí mnoho výhod, protože framework poskytuje funkce, které jsou již naprogramovány, a můžeme se na ně spolehnout. Tyto funkce jsou již využívány v jiných aplikacích a dobře otestovány, a proto je spolehlivější je využít, nežli psát vlastní, méně otestovaný kód. Díky frameworku můžeme tedy zvýšit spolehlivost celé aplikace.

Mezi další výhody MVC Frameworků patří:

- Framework je již hotová část funkční aplikace, proto při použití dostaneme spustitelnou aplikaci
- Oddělují prezenční vrstvu od datové, a proto je snadnější úprava prezenční vrstvy, protože je mnohem přehlednější a lépe udržovatelná
- Velkou výhodou je i paralelní vývoj jednotlivých vrstev

- Při práci s modelem můžeme definovat rozhraní a s ním pak mohou pracovat i ostatní vrstvy, a to bez znalostí jeho implementace. Tato možnost usnadňuje především vývoj prezentační vrstvy
- Snadnější rozšiřování aplikace díky jednotlivým vrstvám

Architektura MVC rozděluje aplikaci do jednotlivých modulů, díky čemuž se aplikace stává spolehlivější, její vytváření je jednodušší, snižují se počty chyb vzniklých při jejím vývoji, což také znamená snížení nákladů na vytvoření aplikace. Další velkou výhodou je i znovupoužitelnost kódu jednotlivých částí aplikace, a zároveň snadnější zvětšování, úprava, udržování i oprava chyb.

Možné nevýhody MVC Frameworků:

- Pokud chceme využívat MVC framework při vývoji webové aplikace, musíme jej nejprve prostudovat a naučit se ho používat. Zvládnutí určitého frameworku může být velmi obtížné, avšak po jeho zvládnutí vývoj webové aplikace velmi usnadňuje
- Další nevýhodou může být i to, že framework je součástí naší aplikace, a proto je nutné k aplikaci přidat i jeho knihovny
- Při použití frameworku vzniká poměrně silná vazba rámce na aplikaci, protože framework se skládá ze skupin tříd, abstraktních tříd a rozhraní, které v aplikaci musíme implementovat nebo rozšiřovat. Dnešní frameworky se snaží tuto závislost snížit na minimum
- Výkon aplikace může být ovlivněn frameworkem, protože ten si vytváří hodně pomocných objektů [20] [21]

3.3 Model – view – controller frameworky pro vývoj webových aplikací

V této kapitole budu popisovat specifické vlastnosti MVC frameworků, které jsou naprogramovány v jazyce Java a tento jazyk využívají i při vývoji aplikace.

Frameworky, kterými se budu v této kapitole zabývat, jsou Struts 2, Jboss Seam, Apache Wicket, Apache Tapestry.

3.3.1 Struts 2

Apache Struts je open-source webový framework, který je založený na jazyce Java. Tento framework původně vytvořil programátor Craig R. McClanahan a v roce 2002 jej převzala společnost Apache Software Foundation, která jej šíří pod licencí Apache Licence 2.0 stejně jako Spring.

Struts 2 je vynikající framework především pro snadný vývoj webových aplikací. Organizuje JSP a servlety založené na HTML formátu a Java kódu. Kolem tohoto frameworku, který má bohatou historii, existuje velmi široká komunita, a proto je velmi často využíván.

[20] Url: http://java.sun.com/blueprints/patterns/MVC-detailed.html. [cit. 1. 4. 2011].

[21] VONDROUŠ, J. *Diplomová práce – Renovace MVC*. 4 – 5 s.

Vlastnosti Struts 2:

- Odstraňuje chyby, které byly v předchozí verzi Struts 1, a tím činí tento framework kvalitnějším a jednodušším
- Struts 2 využívá návrhový vzor Inversion of Control, který známe ze Springu, a díky němu jsou zde zjednodušené akce. Akce totiž nerozšiřují třídy ani rozhraní, které by byly závislé na frameworku, a proto může být akcí i POJO objekt, u kterého je metoda execute vracející datový typ String
- Nemusí rozšiřovat abstraktní třídy, které jsou závislé na frameworku, ale implementuje pouze rozhraní
- Pro výrazy v tomto frameworku může být použit jazyk OGNL (Object Graph Notation Language). Tento jazyk se zde používá i pro typovou konverzi, a proto je tato konverze možná pro základní datové typy a objektové datové typy
- Od verze Java 5 jsou v tomto jazyce zavedeny anotace, které právě mohou být ve Struts 2 použity, a to například pro konfiguraci, kterou již nemusíme psát do XML nebo properties souborů. Anotace lze použít také pro validaci formulářů
- Díky velmi snadné integraci do Spring frameworku, můžeme lehce využít akcí ve Springu, a to přidáním pouze JavaBeanu Spring Frameworku
- Struts 2 má v sobě integrovány značky, které používají AJAX, proto je možná validace dat na straně klienta, obnovení pouze části HTML stránky, automatické doplňování apod.

Shrnutí vlastností:

- Práce výhradně s POJO objekty
- Velmi dobrá integrace Springu a úzká spolupráce s ním
- Pro zobrazení view je možné použít různé technologie, například JSP, JSF, Freemarker nebo Velocity
- Pro prácí s databází můžeme použít nástroje jako jsou JDBC, Hibernate a další
- Podpora stavovosti v některých objektech (checkbox) [22] [23]

Srovnání Struts 2 a Spring MVC Framework:

- Struts 2 je čistě webový framework, zatímco Spring MVC framework je podmnožinou Spring Frameworku
- Spring patří mezi „lightweight“ frameworky, Struts je „heavyweight“
- Spring framework využívá IoC kontejner a podporuje aspektově orientované programování (AOP)

3.3.2 Jboss Seam

Jboss Seam je webový framework s otevřeným zdrojovým kódem (open-source) pro vývoj webových aplikací v jazyce Java. Integruje v sobě další frameworky, jako jsou například asynchronní JavaScript s XML (AJAX), Enterprise Java Beans (EJB 3.0), Java Persistance (JPA) nebo pro prezentační vrstvu JSF (JavaServer Faces). Autorem tohoto framewroku je programátor Gavin King, vedoucí specifikační komise pro WebBeans a autor známého objektově relačního nástroje Hibernate.

Seam se snaží vyřešit problém bezstavového protokolu HTTP pomocí správy stavovosti (Stateful Management), a proto je postaven na stavových EJB komponentech (Stateful EJB Session Beans). Tyto komponenty uchovávají kompletní stav aplikace namísto ukládání do

[22] Url: http://j2eefolks.blogspot.com/2007/04/struts-2-features-and-short-summary.html. [cit. 1. 4. 2011].

[23] Url: http://struts.apache.org/2.2.1.1/index.html. [cit. 1. 4. 2011].

HTTP session nebo databáze. Využitím stavových komponent nenastávají problémy vzniklé použitím HTTP session a přístup do databáze je minimální během jedné konverzace (viz Konverzace v Seam).

K zobrazení prezenční vrstvy se výhradně používá Facelets, což je open-source webový framework. Struktura Facelets dokumentu je obyčejný XHTML kód, který používá tagy různých knihoven (viz tabulka knihovny používané ve Facelets). Pokud chceme tyto knihovny využívat, musíme nejprve deklarovat příslušný jmenný prostor (name space) a jeho prefix.

Konverzace v Seam

Konverzace je základem stavového přístupu při tvorbě webové aplikace a zároveň řeší problém, kdy uživatel se snaží pomocí několika požadavků (či stránek) splnit jeden případ užití nabízený aplikací.

Konverzace mají přitom svůj vlastní životní cyklus, který sice obvykle trvá pouze řádově několik minut, avšak jednotlivé konverzace jsou ukládány odděleně.

V Seamu se využívají dva druhy konverzací, krátkodobá konverzace, která udržuje stav aplikace pouze přes dvě stránky (tj. dva požadavky), např. zadání dat na jedné stránce, zpracování a následné zobrazení na stránce jiné. Tato konverzace je nastavena standardně. Dalším druhem konverzace je dlouhodobá konverzace, která je nezávislá na počtu požadavků (request), a pokud ji chceme začít využívat, musíme nejprve převést krátkodobou konverzaci na dlouhodobou, a poté ji ukončit. Metody, které spouštějí dlouhodobou konverzaci nebo ji ukončují, se označují pomocí anotací @Begin a @End.

Vlastnosti a výhody Seam frameworku:

- Seam využívá frameworky EJB 3.0 a JSF
- Stará se o správu transakcí a výjimek
- Poskytuje možnost autonomní funkce aplikace ve dvou záložkách jednoho prohlížeče
- Umožňuje využití CRUD (vytvoř, získej, aktualizuj, smaž) nad entitami za pomoci tříd a komponent v tzv. Seam CRUD Application Framework
- Umožňuje správu stavovosti aplikace pomocí Stateful Managementu
- Podporuje Facelets pro zobrazení prezentační vrstvy
- Nabízí snadné generování PDF dokumentů, emailů a formátovaného textu (Rich text)
- Umožňuje snadné využití Hibernate Validator

Srovnání Seam frameworku a Spring MVC frameworku:

- Spring framework pracuje s POJO objekty, zatímco Seam využívá EJB komponenty
- Konfigurace v Seam frameworku za použití pouze anotací, ve Spring frameworku můžeme použít také anotace nebo konfiguraci pomocí XML
- Seam framework umožňuje správu stavovosti

Nevýhodou Seam frameworku může být porušení architektury vrstev Java EE aplikace a složité vytváření „kostry aplikace“, ovšem, pokud nepoužijeme vývojové prostředí, které ji vytvoří (např. sada pluginů pro Eclipse IDE z JBoss Tools) [24] [25]

3.3.3 Apache Wicket

Wicket patří mezi odlehčené (lightweight) a komponentově orientované webové frameworky pro programovací jazyk Java. Tento framework vytvořili programátoři Janathan Locke a Miko Matsumura v roce 2004. Od roku 2007 je tento framework šířen pod licencí Apache Licence 2.0, a proto se pojmenoval Apache Wicket. Vývoj aplikací v tomto rámci se podobá vývoji desktopových aplikací, protože programátor nepracuje přímo s protokolem HTTP. Při psaní komponent se Wicket dále velmi podobá stavovým GUI (Graphic user interface) frameworkům, například Java Swing (viz Obrázek 6).

Hlavní výhody Apache Wicket frameworku:

- Veškerý kód je psán v jazyce Java, který se velmi podobá Swingu
- Konfigurace pomocí XML souborů je minimální
- Každá komponenta má svůj vlastní model, který reprezentuje stav komponenty
- Wicket přesně nedefinuje žádnou ORM vrstvu, proto aplikace využívá například Hibernate objekty, EJB nebo POJO objekty jako model
- Prezenční vrstva je tvořena XHTML šablonou, od které je oddělena bussines logika aplikace, která je napsaná v jazyce Java
- Propojení prezenční vrstvy a logiky aplikace je zajištěno pomocí atributu wicket:id, který se nachází u dynamického elementu
- Snadná integrace s jinými frameworky – Hibernate, Spring
- Funkční tlačítko „zpět“ v prohlížeči - stav stránek je ukládán do mapy stránek pro každé okno prohlížeče v uživatelské relaci
- Komponenty naprogramovány ve Wicket mohou být znovu využitelné
- Možnost použití komponenty Panel, která umožňuje vytvořit znovu použitelný celek
- Snadné použití již předpřipravených komponent pro validaci vstupních dat, stránkovaní, apod.
- Pro prohlížeče, které nepodporují JavaScript je možné použít AJAX s alternativou klasického HTTP dotazu
- Kód je standardně bezpečný (zajištěna maximální typová bezpečnost Java kódu) [26] [27]

Srovnání Apache Wicket frameworku a Spring MVC frameworku:

- Wicket je komponentově orientovaný framework, zatímco Spring je požadavkově orientovaný framework
- Wicket nevyužívá stránky JSP ani žádné šablonovací frameworky (například Velocity), Spring JSP využívá a také má možnost použít šablonovací framework
- Možnost vytvoření znovu použitelného celku například pomocí komponenty Panel

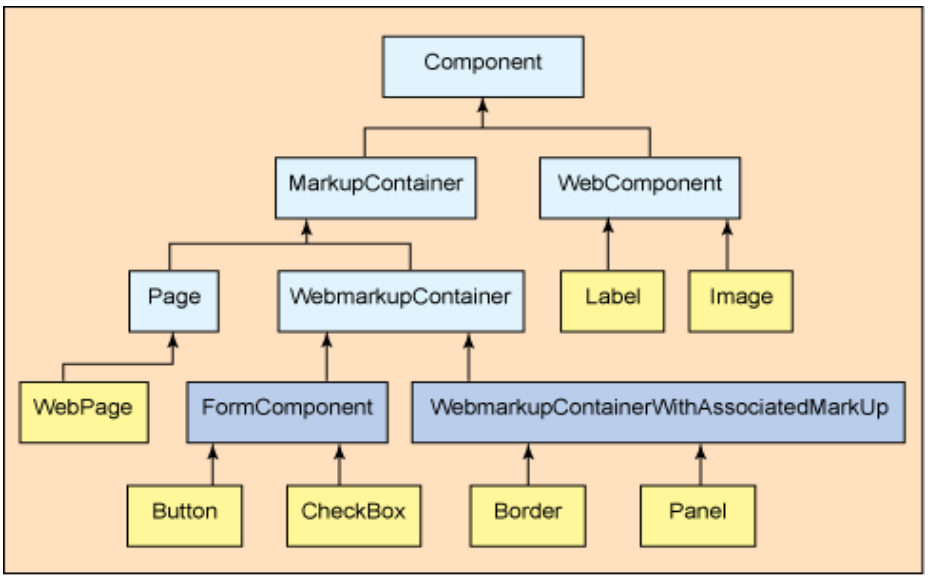
[24] Url: http://docs.jboss.com/seam/latest/reference/en-US/html/Book-Preface.html. [cit. 2. 4. 2011].

[25] Url: http://www.kubasek.cz/blog/2008/02/17/predstaveni-aplikacniho-frameworku-jboss-seam/. [cit. 2. 4. 2011].

[26] Url: http://wicket.apache.org/meet/features.html. [cit. 2. 4. 2011].

[27] Url: http://wicket.apache.org/meet/introduction.html. [cit. 2. 4. 2011].

Obrázek 6 Komponenty Apache Wicket



Zdroj: http://www.ibm.com/developerworks/web/library/wa-aj-wicket/?S_TACT=105AGY82&S_CMP=GENSITE.

3.3.4 Apache Tapestry

Apache Tapestry je open-source komponentově orientovaný framework pro tvorbu webových aplikací v jazyce Java. Tento framework vytvořil programátor Howard Lewis Ship a později jej převzala korporace Apache Software Foundation a nyní je rozšířen pod licencí Apache 2.0 Licence.

Apache Tapestry je založen na běžném Java Server API, které rozšiřuje. Každá webová aplikace je složena z několika stránek, a tyto stránky jsou poté složeny z různých komponent. Využití komponent zlepšuje výrazně produktivitu vývoje webové aplikace a toto je i důvod proč nové webové frameworky začaly komponenty využívat (ASP.NET, JavaServer Faces). Apache Tapestry dále může využívat i jiné frameworky například Spring, Hibernate a jiné.

Části Tapestry frameworku:

Jádro – to obsahuje základní složky frameworku, a zároveň i další třídy pro využívání různých služeb na webových stránkách a v komponentách.

Vložené komponenty – obsahují výkonné add-on komponenty, které poskytla veřejnost (například komponenty pro sestavení HTML odpovědi k interpretaci webových stránek).

Třídy pro podporu anotací – tato knihovna obsahuje anotace, které můžeme využít pro vytvoření různých operací přímo v Java kódu, bez nutnosti specifikace těchto operací na webové stránce nebo v komponentě. Anotace je možné využívat od JDK 1.5.

Podpora pro portlety – je zajištěna pomocí add-on modulu, díky kterému můžeme vytvářet JSR 168 portlety.

Hlavní vlastnosti Apache Tapestry:

- Framework od své verze 5 využívá Tapestry Inversion of Control, který zajišťuje větší jednoduchost a sílu frameworku, a to bez používání XML
- Odpadá nutnost konfigurace pomocí XML souborů
- Každá webová stránka aplikace je složena z komponent
- Snadná integrace s jinými frameworky – Spring, Hibernate
- Možnost využití anotací
- Nabízí i využití návrhového vzoru Singleton [28]
- Pokud využijeme jmennou konvenci, musí každá metoda začínat slovem „on“, poté je nutno uvést název události, dále pak slovo „from“ a id komponenty, ke které se událost vztahuje – například název formuláře je LoginForm a událost bude onSubmit, takže název metody bude vypadat takto – onSubmitFromLoginForm
- Využívá strategii Page Pooling – pokud je odeslána žádost o zobrazení webové stránky, IoC kontejner zjistí, zda je v poolu stránek (tj. zda je zde její instance)
- Možné spuštění aplikace v debug módu a snadnější ladění aplikace za použití IoC kontejneru. [29] [30] [31]

Srovnání Apache Tapestry frameworku a Spring MVC frameworku:

- Tapestry je komponentově orientovaný framework, zatímco Spring je požadavkově orientovaný framework
- Tapestry i Spring využívají Inversion Of Control
- Tapestry se snadno integruje do Spring frameworku

4 Specifikace požadavků pro webovou aplikaci

Tato kapitola popisuje můj úkol v bakalářské práci, vytvořit aplikaci pro zaznamenávání a publikování sportovních výsledků a jiných novinek k tomuto tématu. Zadaní je technicky řešeno jako webová aplikace na platformě Java 2 Enterprise Edition, s využitím Spring MVC frameworku. Jako webový server je používán Apache Tomcat. Na této aplikaci se snažím ukázat výhody a ulehčení práce při použití tohoto frameworku. Snažil jsem se vytvořit webovou aplikaci pro uživatele přehlednou, snadno použitelnou a hlavně účelnou.

4.1 Základní popis aplikace

[28] Url: http://www.javaworld.com/javaworld/jw-04-2003/jw-0425-designpatterns.html. [5. 4. 2011].

[29] Url: http://tapestry.apache.org/introduction.html. [5. 4. 2011].

[30] Url: http://tapestry.apache.org/tapestry5/. [5. 4. 2011].

[31] Url: http://tapestry.apache.org/tapestry5/guide/lifecycle.html. [5. 4. 2011].

Vytvořená webová aplikace slouží ke sledování sportovních výsledků, čtení sportovních článků, psaní komentářů k jednotlivým článkům a dále poskytuje i jiné informace z oblasti sportu, například - historie jednotlivých sportů, týmové sestavy a soupisky hráčů. Jednotlivé sporty jsou rozděleny na týmové a netýmové.

Publikovat nové výsledky a informace je v této aplikaci oprávněn pouze administrátor, který má i své speciální webové rozhraní. Pro přístup do aplikace je nutná autorizace uživatele nebo jeho registrace, pokud se jedná o uživatele nového (viz kapitola Přístup do aplikace).

4.2 Proč vytvářet novou aplikaci

Aplikací pro sledování sportovních výsledků jsou na internetu desítky, ale většina z nich poskytuje pouze výsledky ale už ne informace o jednotlivých sportech. Tyto aplikace jsou bohužel podle mého názoru většinou nepřehledné a uživatel se v nich těžce orientuje. Další nevýhodou je fakt, že většina dostupných webových aplikací je určena hlavně pro sázkaře a proto se zde zobrazují kursy jednotlivých sázkových kanceláří, což nemusí každého zajímat.

Proto jsem se ve své aplikaci zaměřil spíše na skupinu sportovních nadšenců, která preferuje aktuální informace ze světa sportu a není orientována jen na výsledky kvůli sázkařským kurzům. Uživateli této aplikace by tedy měli být běžní zájemci o sport a příznivci sportovních aktivit.

4.3 Funkční požadavky aplikace

V této kapitole popisuji základní funkční požadavky webové aplikace. Uvádím zde pouze některé požadavky, které se vztahují k běžnému uživateli a administrátorovi.

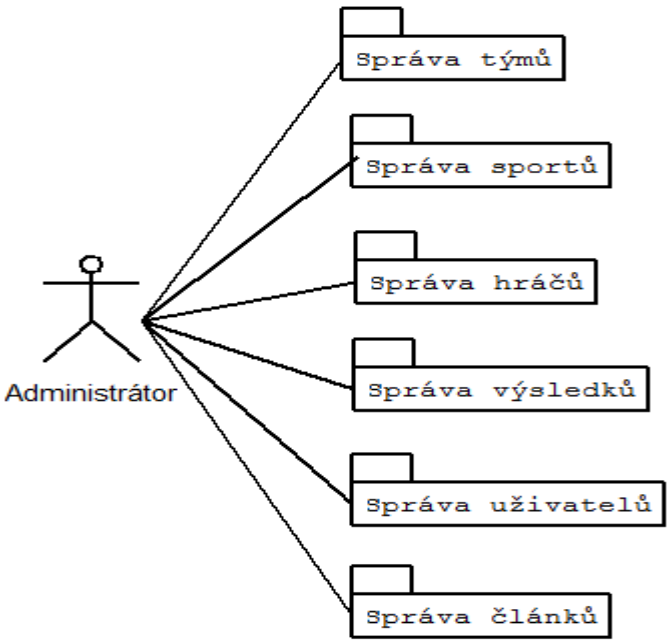
1. Uživatel se přihlašuje do systému
2. Uživatel se registruje do systému
3. Uživatel prohlíží články, které se týkají daného sportu
4. Uživatel prohlíží výsledky jednotlivých zápasů
5. Uživatel prohlíží týmové sestavy
6. Uživatel prohlíží soupisky hráčů
7. Uživatel přidává příspěvek do diskuse ke článku
8. Uživatel prohlíží historii vybraného sportu
9. Administrátor se přihlašuje do systému
10. Administrátor vytváří, edituje a maže články, které se týkají daného sportu
11. Administrátor vytváří, edituje a maže výsledky jednotlivých zápasů
12. Administrátor vytváří, edituje a maže týmové sestavy
13. Administrátor vytváří, edituje a maže soupisky hráčů
14. Administrátor vytváří, edituje a maže sporty

4.4 Diagramy užití

Diagram případů užití (Use Case Diagram) se používá k popisu chování systému z hlediska uživatele. Tuto specifikaci jsem vytvořil tak, že jsem systém rozdělil do logických

celků a tyto celky zakreslil do diagramu pomocí balíčků (packages). Každý balíček jsem poté rozkreslil do menších diagramů, na kterých již ukazuji příslušné případy užití.

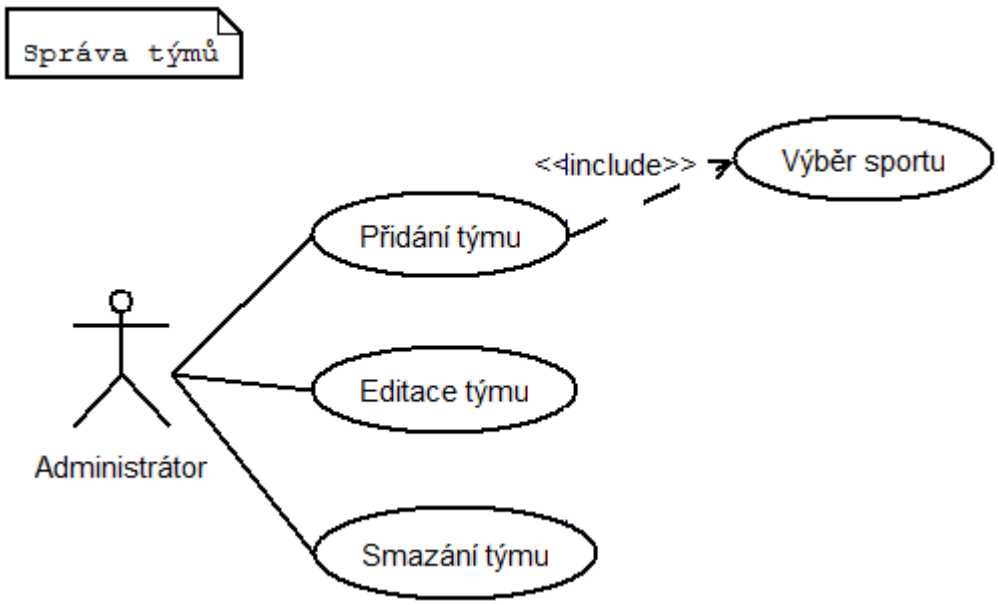
Obrázek 7 Rozdělení systému do logických celků



Správa týmů:

Pomocí diagramu užití zde popisuji logický celek správa týmů. Tento celek zahrnuje případy užití přidání týmů, smazání týmů a editace týmů. V tomto diagramu užití vystupuje pouze role administrátora.

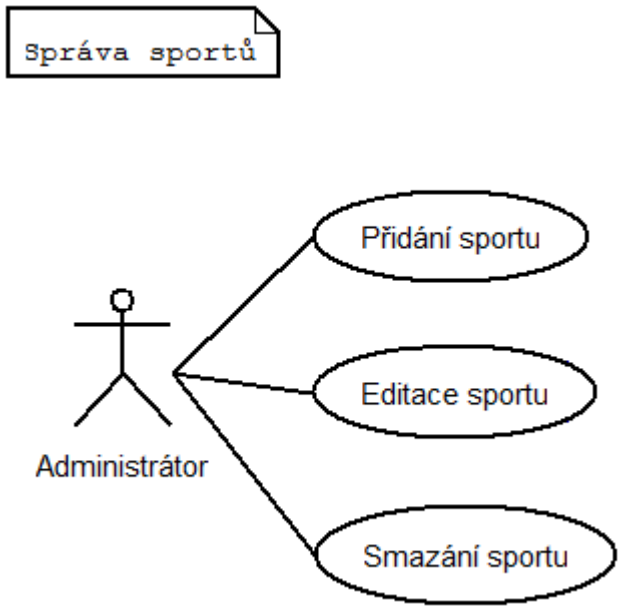
Obrázek 8 Diagram správa týmů



Správa sportů:

Pomocí diagramu užití je zde popsán logický celek správa sportů. Tento celek zahrnuje případy užití přidání sportu, smazání sportu a editaci sportu. V tomto diagramu užití vystupuje pouze role administrátora.

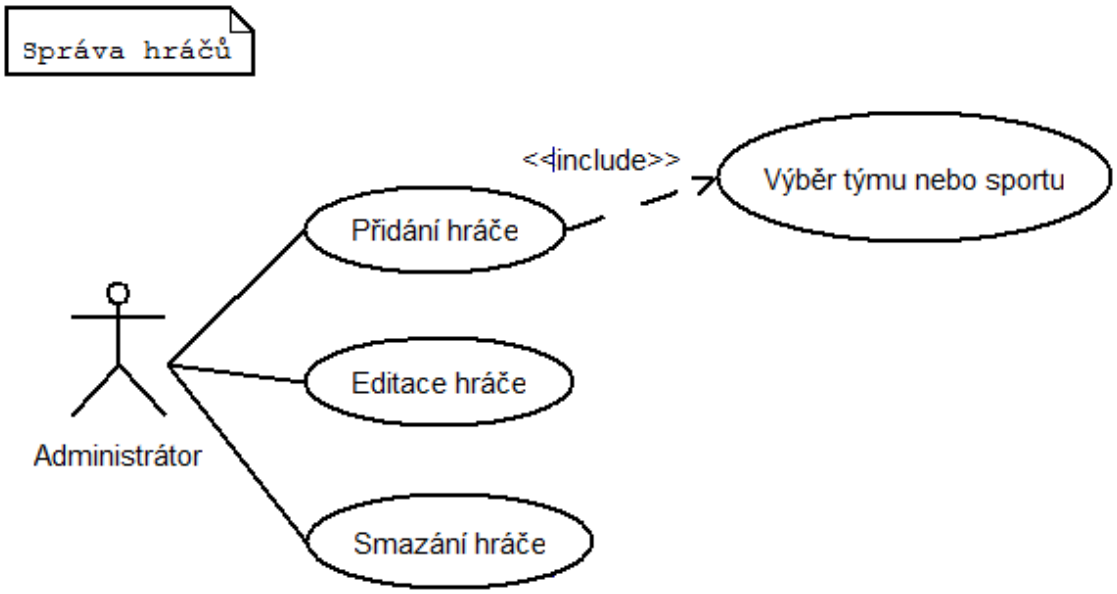
Obrázek 9 Diagram správa sportů



Správa hráčů:

Pomocí diagramu užití je zde popsán logický celek správa hráčů. Tento celek zahrnuje případy užití přidání hráče, smazání hráče a editaci hráče. V tomto diagramu užití vystupuje pouze role administrátora.

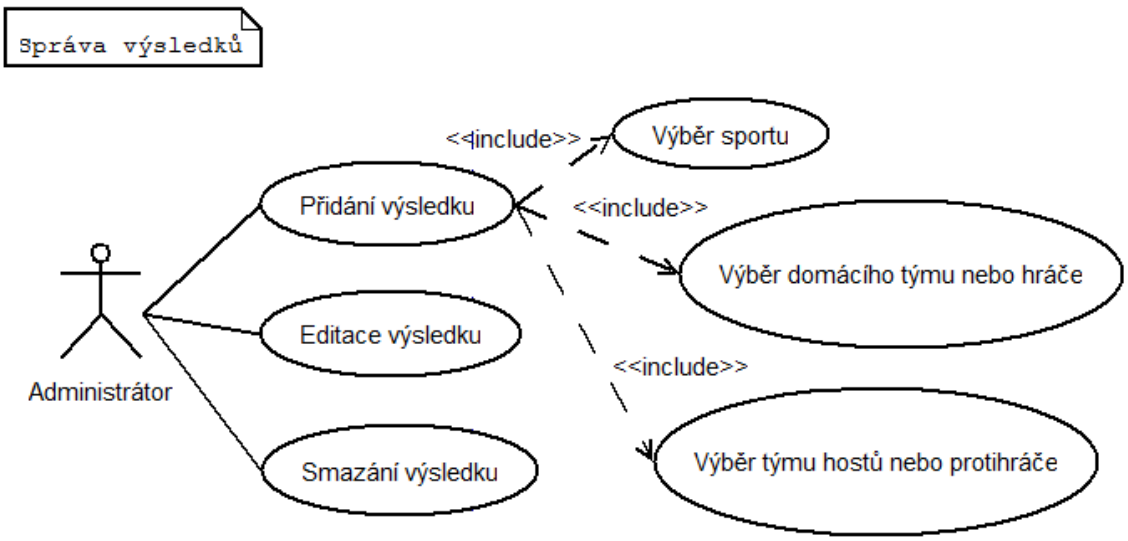
Obrázek 10 Diagram správa hráčů



Správa výsledků:

Pomocí diagramu užití je zde popsán logický celek správa výsledků. Tento celek zahrnuje případy užití přidání výsledku, smazání výsledku a editaci výsledku. Při přidávání výsledku je nutné zvolit tým domácích hráčů a tým hostů nebo hráče a protihráče. Tato volba závisí na druhu sportu, ke kterému se výsledek vztahuje. V tomto diagramu užití vystupuje pouze role administrátora.

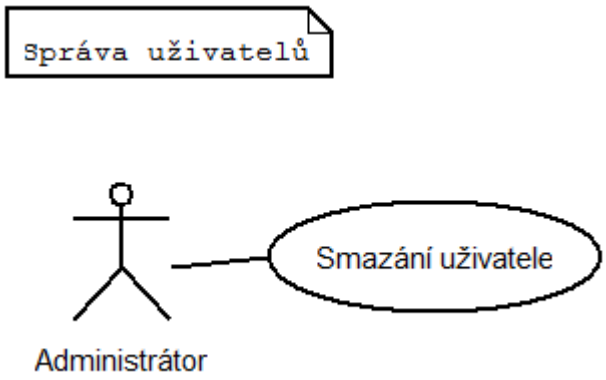
Obrázek 11 Diagram správa výsledků



Správa uživatelů:

Pomocí diagramu užití je zde popsán logický celek správa uživatelů. Tento celek zahrnuje jediný případ užití smazání uživatele. V tomto diagramu užití vystupuje pouze role administrátora.

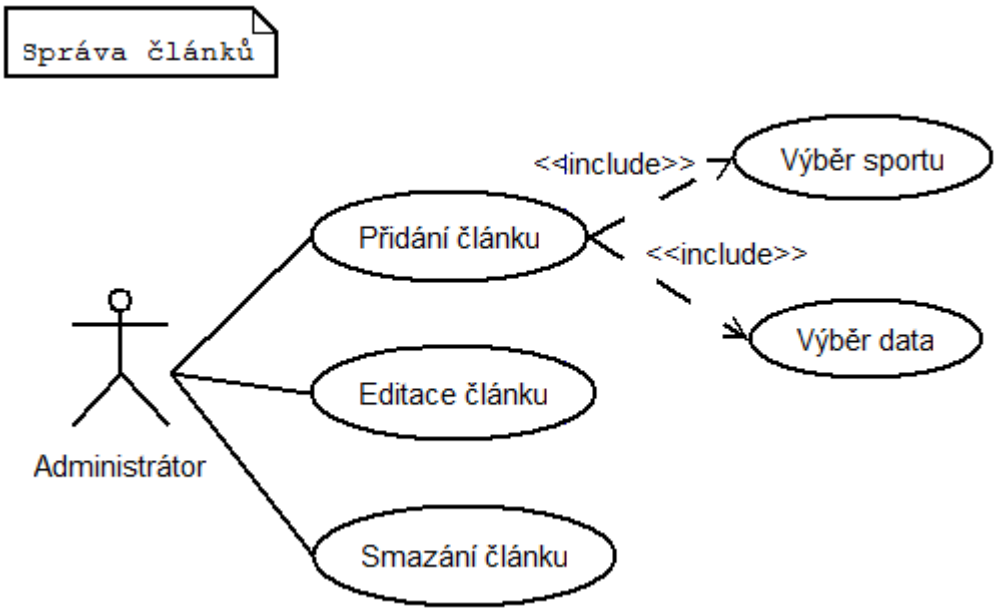
Obrázek 12 Diagram správa uživatelů



Správa článků:

Pomocí diagramu užití je zde popsán logický celek správa článků. Tento celek zahrnuje případy užití přidání článku, smazání článku a editaci článku. V tomto diagramu užití vystupuje pouze role administrátora.

Obrázek 13 Diagram správa článků



5 Analýza a návrh webové aplikace

V této kapitole nejprve popisuji analýzu webové aplikace, ve druhé části kapitoly se zabývám konkrétním řešením webové aplikace. Součástí kapitoly je datová analýza a funkční analýza.

5.1 Datová analýza webové aplikace

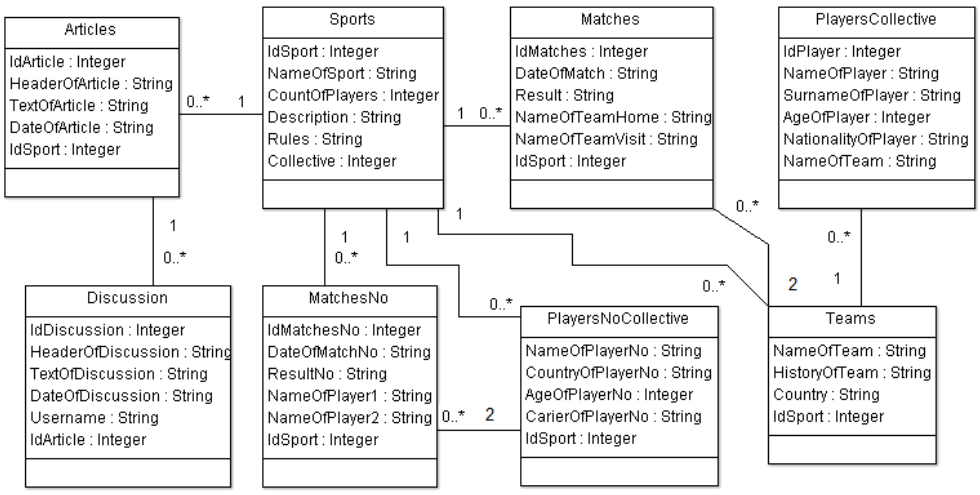
Datová analýza se skládá z diagramu tříd, lineárního zápisu typů entit a datového slovníku.

5.1.1 Diagram tříd

Diagram tříd je zobrazení statické struktury prostřednictvím tříd a vztahů mezi nimi.

Třídní diagram webové aplikace pro zaznamenávání a publikování sportovních výsledků a jiných novinek je zobrazena na obrázku 5. V tomto diagramu jsou zobrazeny pouze třídy a parametry. Metody, které jsou obsaženy v jednotlivých třídách, v tomto diagramu nejsou. Kompletní diagram tříd je v příloze č. 1.

Obrázek 14 Diagram tříd webové aplikace



5.1.2 Lineární zápis

Lineární zápis (textový) popisuje jednotlivé entity a vztahy mezi těmito entitami. Podrobnější specifikace je popisována v datovém slovníku (Příloha číslo 2).

Articles (IdArtcile, HeaderOfArticle, TextOfArticle, DateOfArtcile, IdSport)

Discussion (IdDiscussion, HeaderOfDiscussion, TextOfDiscussion, DateOfDiscussion, Username, IdArtcile)

Matches (IdMatches, DateOfMatch, Result, NameOfTeamHome, NameOfTeamVisit, IdSport)

MatchesNo (IdMatchesNo, DateOfMatchNo, ResultNo, NameOfPlayer1, NameOfPlayer2, IdSport)

PlayersCollective (IdPlayer, NameOfPlayer, SurnameOfPlayer, AgeOfPlayer, NationalityOfPlayer, NameOfTeam)

PlayersNoCollective (NameOfPlayer, CountryOfPlayer, AgeOfPlayer, CarierOfPlayer, IdSport)

Sports (IdSport, NameOfSport, CountOfPlayers, Description, Rules, Collective)

Teams (NameOfTeam, HistoryOfTeam, Country, IdSport)

Legenda: primární klíč, cizí klíč

Vysvětlení významu jednotlivých entit:

Articles – v této tabulce jsou záznamy o jednotlivých sportovních článcích.

Discussion – v této tato tabulce jsou uloženy příspěvky diskuse.

Matches – v této tabulce jsou uloženy výsledky jednotlivých zápasů pro týmové sporty.

MatchesNo – v této tabulce jsou uloženy výsledky jednotlivých zápasů pro netýmové sporty

PlayersCollective – v této tabulce jsou uloženi jednotliví hráči týmů.

PlayersNoCollective – v této tabulce jsou uloženi hráči netýmových sportů.

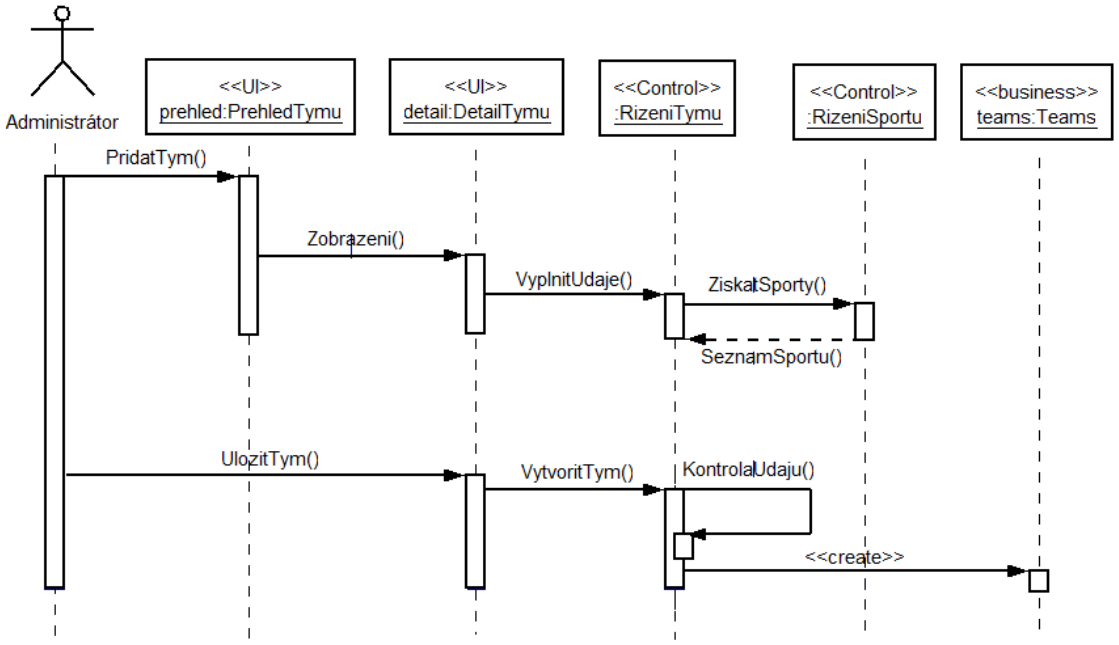
Sports – v této tabulce jsou uloženy jednotlivé sporty.

Teams – v této tabulce jsou uloženy jednotlivé týmy.

5.2 Funkční analýza

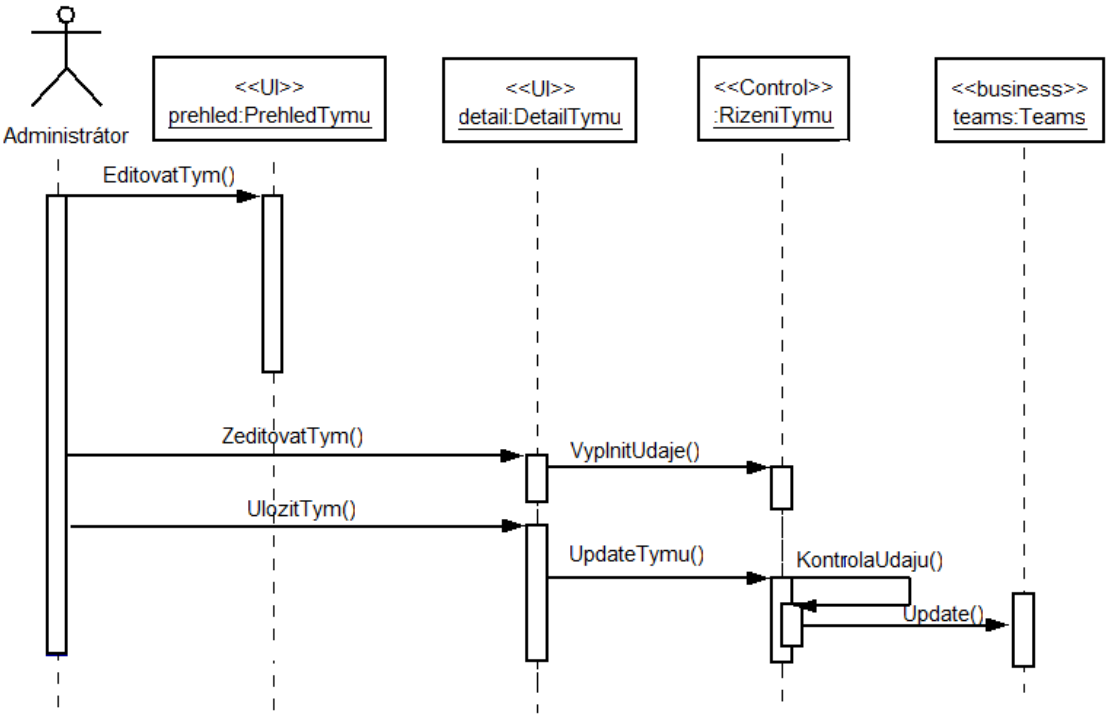
V této kapitole popisuji pomocí sekvenčního diagramu konkrétní případ užití. Pro ukázku jsem vybral případy užití - přidaní týmu, editaci týmu a smazání týmu.

Obrázek 15 Sekvenční diagram případu užití – Přidání týmu



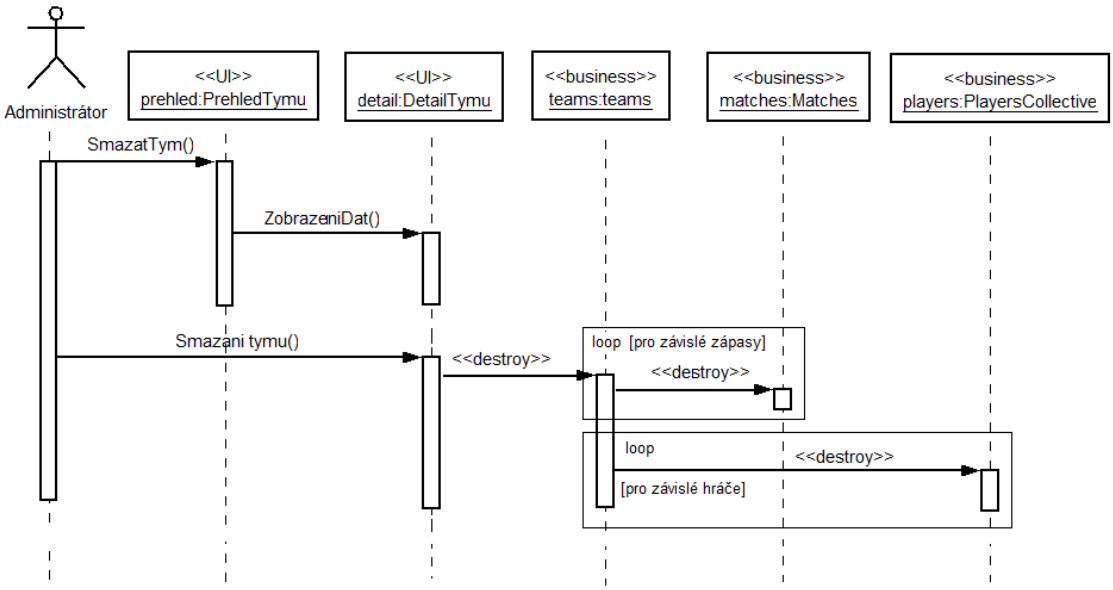
Administrátor je přihlášený do systému a zvolí administraci týmů. Systém poté zobrazí formulář pro přidání nového týmu a administrátor vyplní všechny povinné údaje. Při vyplňování údajů musí administrátor vybrat sport, ke kterému se dány tým vztahuje. Seznam sportů získá systém automatiky. Administrátor vybere volbu uložení týmu a aplikace provede ověření zadaných dat. Pokud nenalezne žádnou chybu, tak systém uloží nový tým. Pokud systém nalezne chybu, zobrazí upozornění o chybě a tým není uložen.

Obrázek 16 Sekvenční diagram případu užití – Editace týmu



Administrátor je přihlášený do systému a zvolí administraci týmů. Systém poté zobrazí formulář pro přidání nového týmu a výpis všech týmů s možnostmi smazaní týmu a editace týmu. Administrátor zvolí volbu editace týmu. Systém poté zobrazí formulář s hodnotami vybraného týmu a administrátor má možnost hodnoty upravit. Poté systém provede kontrolu zadaných dat, a pokud nenalezne žádnou chybu, aktualizuje tým. Pokud systém nalezne chybu, zobrazí upozornění o chybě a tým není aktualizován.

Obrázek 17 Sekvenční diagram případu užití - Smazání týmu



Administrátor je přihlášený do systému a zvolí administraci týmů. Systém poté zobrazí formulář pro přidání nového týmu a výpis všech týmů s možnostmi smazaní týmu a editaci týmu. Administrátor zvolí volbu smazání týmu a daný tým je poté smazán. Dále systém automaticky smaže všechny závislé objekty, jako jsou výsledky a hráči.

5.3 Návrh systému

Účelem této kapitoly je ukázat a vysvětlit konkrétní způsoby implementace některých problémů pomocí Spring frameworku. Popisuji zde objektově-relační mapování aplikace pomocí Hibernate frameworku, využití návrhového vzoru Inversion of Control a použití frameworku pro autorizaci a autentizaci uživatelů.

5.3.1 ORM – Objektově-relační mapování

Jako nástroj pro objektově – relační mapování jsem si zvolil framework Hibernate, protože k tomuto nástroji je velmi dobře zpracována dokumentace a jeho integrace do Springu je velmi dobře podporována. Jako databázi jsem si pak zvolil MySQL, protože je volně dostupná a má dostatečný výkon.

Integrace Hibernate do aplikace postavené na Spring frameworku:

V této části nejprve popisuji inicializaci Hibernate v konfiguračním souboru dispatcher-servlet.xml. Tato konfigurace je naprostým základem, pokud chceme pracovat s nástrojem Hibernate v aplikaci vytvářené pomocí Spring frameworku.

Příklad konfigurace v dispatcher-servlet.xml:

```
<bean id="DataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
<property name="driverClassName">
<value>com.mysql.jdbc.Driver</value>
</property>
<property name="url">
<value>jdbc:mysql://localhost/bp</value> <!-- volba databaze -->
</property>
<property name="username">
<value>root</value>
</property>
</bean>
```

Bean s názvem DataSource definuje zdroj dat typu *org.springframework.jdbc.datasource.DriverManagerDataSource,* který definuje různé vlastnosti připojení, které jsou důležité pro přístup k databázi. V mé aplikaci se připojuji k MySQL databázi, proto musím použít ovladač databáze MySQL. Vlastnost s názvem *driverClassName* ukazuje na název třídy, kde se nachází ovladač databáze MySQL. Další vlastnost s názvem *url* představuje URL potřebné k připojení k databázi. Poslední dvě vlastnosti jsou *username* a *password,* které představují uživatelské jméno a heslo nutné pro přístup k databázi.

Mapování tabulek databáze a získávání dat z těchto tabulek:

V této části popisuji, jak probíhá mapování tabulek pomocí Spring frameworku a nástroje Hibernate. V druhé části je popisuji získávání dat z těchto tabulek a na konci popisuji nastavení v konfiguračním souboru dispatcher-servlet.xml, které je nutné pro propojení tříd a databázových tabulek.

Mapování tabulek probíhá tak, že si nejprve vytvoříme třídu, která představuje databázovou tabulku, musí implementovat rozhraní Serializable kvůli serializaci atributů a dále také musíme použít anotace. Anotace typu @Entity ukazuje, že se jedná o objekt v databázi. Dále je třeba vytvořit anotaci @Table(name="nazev_tabulky"), která představuje konkrétní tabulku v databázi, na kterou má být třída mapována.

Metody get() jsou anotovány pomocí @Column(name="nazev_atributu"), pomocí které se mapují jednotlivé atributy tabulky. Primární klíč je anotován pomocí @Id. Mapování tabulek lze také provádět v konfiguračním souboru pomocí xml.

Příklad třídy Sports:

```
@Entity
@Table(name="sports")
public class Sports implements Serializable {
        private int idSport;
        @Id
        @Column(name="IdSport")
        public int getIdSport() {
                return idSport;
                }
        //dalsi atributy a ostatni metody set a get (setter, getter)
        }
```

Po vytvoření třídy reprezentující databázovou tabulku, si vytvoříme třídu, která zajišťuje perzistenci objektů v databázi. V této třídě inicializujeme HibernateTemplate a nastavíme SessionFactory (dané sezení).

Příklad třídy SportsDAOImpl:

```
        public class SportsDAOImpl implements SportsDAO {
        private HibernateTemplate hibernateTemplate;
           public void setSessionFactory(SessionFactory sessionFactory) {
              this.hibernateTemplate = new HibernateTemplate(sessionFactory);
           }
           /*
            * Vypis vsech tymovych sportu
            * List<Sports>
            */
           public List<Sports> listOfSports() {
              return hibernateTemplate.find("FROM Sports s WHERE s.collective = 1");
           }
           //dalsi metody napr. update, delete …
        }
```

Aby aplikace fungovala, musíme vytvořit bean v konfiguračním souboru dispatcher-servlet.xml, který se odkazuje na anotovanou třídu Sports.java.

Příklad konfigurace v dispatcher-servlet.xml:

```
<bean id="mySessionFactory3"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
              <property name="dataSource" ref="DataSource" />
              <property name="annotatedClasses">
                 <list>
                    <value>webapp.controller.sport.Sports</value>
                 </list>
              </property>
              <property name="hibernateProperties">
                 <props>
                    <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                    <prop key="hibernate.show_sql">false</prop>
                 </props>
              </property>
           </bean>
```

Bean s názvem *mySessionFactory3* definuje SessionFactoryBean, který je zodpovědný za vytváření Session objektů, přes které se vykonávají transakce a provádí se přístup k datům.

Vlastnost s názvem *dataSource* se odkazuje na zdroj dat, který jsem již definoval. Další vlastnost s názvem *annotatedClasses* definuje svou hodnotou všechny třídy, které jsou anotovány pomocí Hibernate anotací, v tomto případě třída Sports. Poslední vlastnost *hibernateProperties* definuje nastavení Hibernate, jako určení typu databáze (vlastnost *hibernate.dialect*).

Využití HibernateTemplate je velmi efektivní, protože se programátor nemusí starat například o zachytávání výjimek nebo o správu transakcí – toto zajišťuje rámec Spring. HibernateTemplate poskytuje i další metody například save(), update(), delete() a další.

5.3.2 Využití návrhového vzoru Inversion of Control

Návrhový vzor Inversion of Control je jedním ze základních specifik Spring frameworku. Nejčastější způsob řešení IoC je ve Springu použití Setter Injection.

Ve webové aplikaci tento návrhový vzor využívám a budu zde popisovat, jak se řeší způsob Setter Injection.

Základem využití Setter Injection je vytvoření třídy, která bude představovat závislost, v daném případě to bude třída SportsDAOImpl, která implementuje rozhraní SportDAO. Tato závislost bude poté vkládána do třídy SportsDAOController. Zdrojový kód třídy SportsDAOImpl zde nebudu uvádět, protože je uvedený v předchozím příkladě objektově-relačního mapování.

SportsDAOController:

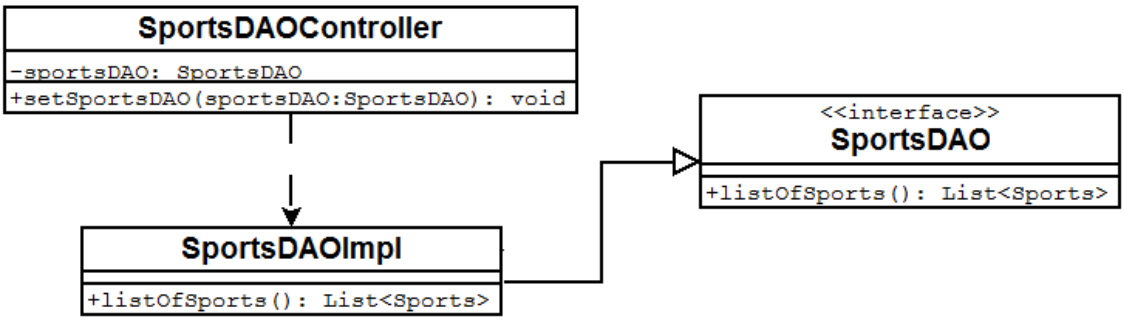
```
import webapp.dao.sports.SportsDAO;
import org.springframework.web.servlet.mvc.multiaction.MultiActionController;
public class SportsDAOController extends MultiActionController {
   private SportsDAO sportsDAO;

   public void setSportsDAO(SportsDAO sportsDAO) {
      this.sportsDAO = sportsDAO;
   }
}
```

Třída SportsDAOController dědí vlastnosti ze třídy MultiActionController. Instance třídy SportsDAOImpl je vkládána pomocí metody setSportsDAO – způsob setter injection.

Třídy a rozhraní používané ve zmíněné webové aplikaci:

- Třída SportsDAOController reprezentuje závislou třídu, závislost je aplikována pomocí třídy SportsDAOImpl
- Rozhraní SportsDAO definuje závislé metody
- Třída SportsDAOImp představuje závislost, která je aplikována ve třídě SportsDAOController

Obrázek 18 Diagram tříd využití Setter Injection



5.3.3 Využití frameworku pro zajištění autorizace a autentizace uživatelů – Spring Security

Tento framework je jeden z nejčastěji používaných nástrojů pro zajištění autorizace a autentizace uživatelů v aplikacích postavených na Spring frameworku ale i na jiných. V mé webové aplikaci využívám scénář řízení přístupu k jednotlivým částem webové aplikace. Spring security ale nabízí i další dva scénáře – řízení přístupu k metodám objektů servisní vrstvy a řízení přístupu k doménovým objektům na základě ACL (Access Control List), tyto jsem ale nevyužil.

Při použití toho frameworku musíme nejprve stáhnout a připojit potřebné knihovny, které nalezneme na oficiálních webových stránkách. [32]

Konfigurace Spring Security:

Základem konfigurace toho frameworku je deklarace filtru DelegatingFilterProxy, který zajišťuje zabezpečení celé webové infrastruktury, v konfiguračním souboru web.xml. Po přidání tohoto filtru je musíme vytvořit aplikační kontext pro další konfiguraci Spring Security.

Klíčovým prvkem v souboru aplikačního kontextu pro Spring Security je prvek <http>. V tomto prvku se definují všechny webové adresy dané aplikace, které mají být přístupné pro určitou roli.

Appliaction-context.xml:

```
<http auto-config='true'>
   <intercept-url pattern="/**" access="ROLE_USER" />
   <http-basic />
</http>
```

Pomocí této deklarace jsme definovali, že pro roli ROLE_USER jsou přístupné všechny webové stránky aplikace. Toto nastavení je pouze základní, avšak v aplikačním kontextu Spring Security můžeme využít mnoho nastavení.

Jedno z nejčastějších nastavení je deklarace prvku <form-login> pomocí kterého můžeme nastavit webovou stránku, která bude spuštěna vždy před přihlášením do aplikace.

[32] http://static.springsource.org/spring-security/site/downloads.html

Pomocí atributu default-target-url definujeme, na kterou webovou stránku jsme po přihlášení přesměrováni.

Příklad přesměrování:

```
<http pattern="/login.htm*" security="none"/>
 <http>
   <intercept-url pattern='/**' access='ROLE_USER' />
   <form-login login-page='/login.htm' default-target-url='/home.htm'
         always-use-default-target='true' />
 </http>
```

V posledním kroku musíme definovat autentizačního poskytovatele, který slouží k vyhledávání uživatelského jména, hesla nebo autority. V tomto případě vyhledává tyto údaje z databáze.

```
<authentication-manager>
   <authentication-provider>
    <jdbc-user-service data-source-ref="DataSource"/>
   </authentication-provider>
 </authentication-manager>
```

Prvek <jdbc-user-service> se odkazuje na zdroj dat s názvem DataSource, který je definován v konfiguračním souboru dispatcher-servlet.xml. Ukázka tohoto souboru i definice zdroje dat je uvedena v kapitole objektově-relačního mapování.

6. Závěr

Ve své bakalářské práci jsem vytvořil aplikaci pro zaznamenávání a publikování sportovních výsledků a jiných novinek k tomuto tématu a poukázal jsem na výhody a ulehčení práce při použití tohoto frameworku. Mým cílem bylo vytvořit přehlednou, snadno použitelnou a rozšířitelnou webovou aplikaci.

Mnou vytvořená webová aplikace slouží jako ukázka práce se Spring frameworkem. Uvědomuji si, že webových aplikací pro sledování sportovních výsledků je na internetu spousta, ale většina z nich poskytuje pouze výsledky, avšak nikoli informace, neboť tyto aplikace jsou určeny hlavně pro sázkaře. Já jsem se zaměřil ve své tvorbě především na skupinu sportovních nadšenců, která preferuje aktuální informace ze světa sportu a není orientována jen komerčně v rámci sázkařských kurzů. Uživateli této aplikace by tedy měli být běžní zájemci o sport a příznivci sportovních aktivit.

Vytvořená aplikace by se dala určitě zdokonalit a rozšířit a to především v tomto směru:

1. Změna databázové vrstvy na jinou velkou databázi například Oracle, což by umožnilo používat například bitmapové indexy nebo indexově organizované tabulky, a tím by vznikla lepší struktura ukládání dat v databázi a zároveň bychom mohli dosáhnout vyššího výkonu aplikace při nasazení do reálného provozu.
2. Zdokonalit by se mohl i design aplikace, a to za pomoci profesionálních designerů.
3. Články by se mohly řadit do různých kategorií, například odborné, naučně-populární, běžné, bulvární apod.
4. Dala by se vytvořit fotogalerie jednotlivých hráčských týmů a zároveň vytvořit i medailonky známých sportovních osobností.

Závěrem bych chtěl dodat, že má práce bude snad i malým přínosem pro všechny zájemce, například z řad studentů, kteří by se rádi dozvěděli něco bližšího o tomto frameworku, který má sice dobře zpracovanou dokumentaci a rozsáhlou doménu uživatelů, avšak v českých pramenech se na toto téma píše jen velmi málo.

Seznam použité literatury

Monografická publikace (knihy):

[1]. JOHNSON, R. *Expert One-on-One J2EE Design and Development.* 1st ed. Wrox publishing, 2002. ISBN 0-7645-4385-7.

[2]. HARROP, R., MACHACEK, J. *Pro Spring.* 1st ed. Apress publishing, 2005. ISBN 1-59059-461-4.

[3]. VONDROUŠ, J. *Diplomová práce – Renovace MVC frameworků*.

Elektronické záznamy:

[4]. *TheServerSide.com, Your Enterprise Java Community* [online]. 2000 – 2011. Url: <http://www.theserverside.com/>.

[5]. *Katedra informatiky, Fakulta elektrotechniky a informatiky, VŠB-TUO* [online]. 2011. Url: <http://www.cs.vsb.cz/>.

[6]. *Interval.cz* [online]. 2011. Url: <http://interval.cz/>.

[7]. *BlackWasp* [online]. 2006-2011. Url: <http://www.blackwasp.co.uk/>.

[8]. *Rising sun* [online]. 2011. Url: <http://sqdw.signaly.cz/>.

[9]. *Springsource, community* [online]. 2011. Url: <http://www.springsource.org/>.

[10]. *Moro Systems, Černý kůň vývoje software* [online]. 2006-2010. Url: <http://morosystems.cz/>.

[11]. *Oracle* [online]. 2011. Url: <java.sun.com>.

[12]. *Lets Fun Java, Lets Make Java A World Of Fun* [online]. 2009. Url: <http://j2eefolks.blogspot.com/>.

[13]. *The Apache Software Foundation* [online]. 2000-2011. Url: <http://struts.apache.org/>.

[14]. *JBoss Community* [online]. 2011. Url: <http://www.jboss.org/>.

[15]. *Lukáš Kubásek, Java&Life Enthusiast* [online]. 2008. Url: <http://www.kubasek.cz/>.

[16]. *Apache Wicket* [online]. 2010. Url: <http://wicket.apache.org/>.

[17]. *Developer works* [online]. 2011. Url: <http://www.ibm.com/developerworks/>.

[18]. *Info World, Java World, Solutions For Java Developers* [online]. 2006– 2011. Url: <http://www.javaworld.com/>.

[19]. *Apache Tapestry 5, Code less, deliver more.* [online]. 2006- 2009. Url: <http://tapestry.apache.org/>.

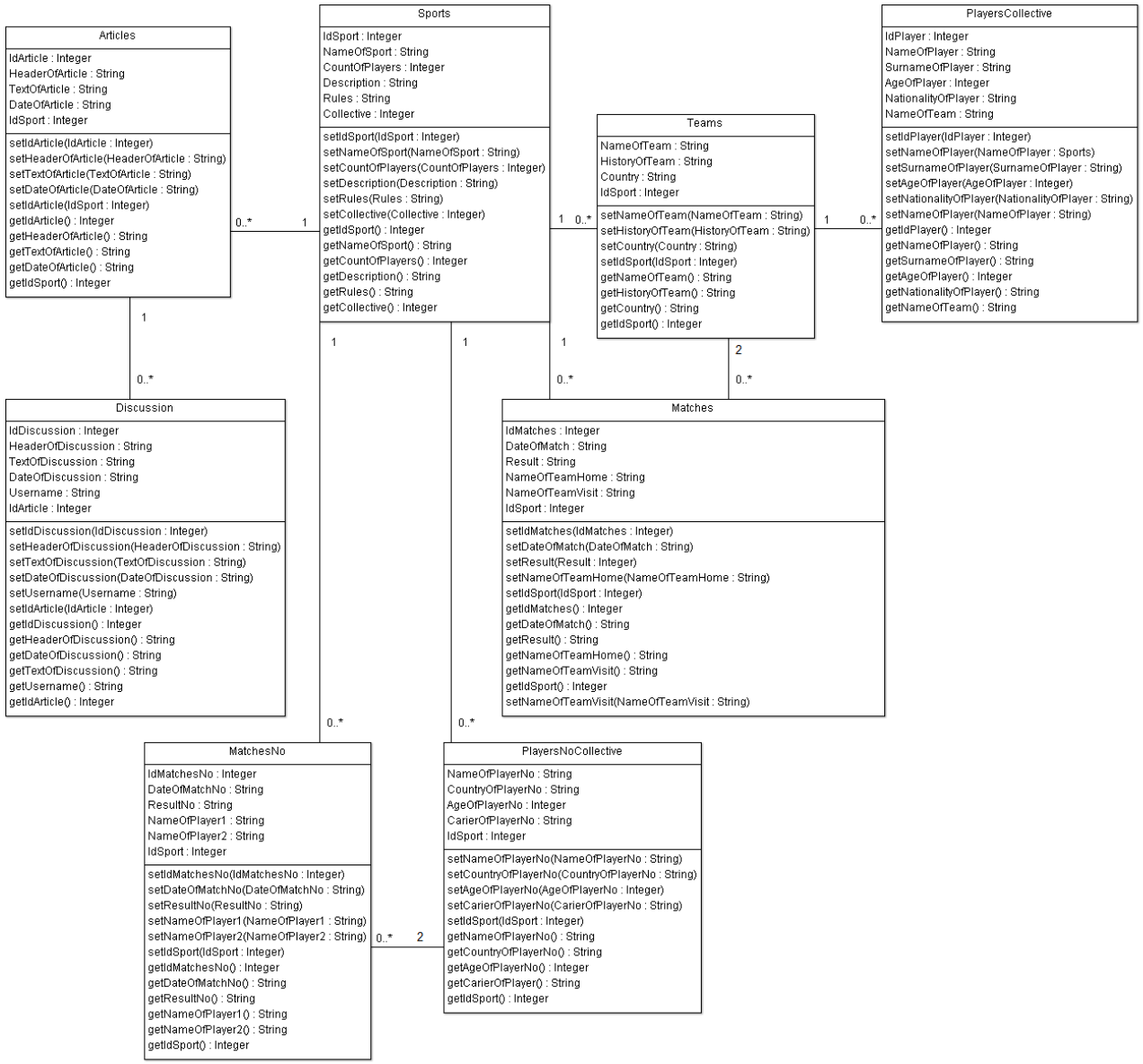
Seznam tabulek

Seznam obrázků

Seznam příloh

Přílohy

Příloha 1



Příloha 2

Datové slovníky entit

Tabulka Sports

Název atributu	Datový typ	Integritní omezení	Index	Popis
IdSport	int	-	primary	Primární klíč
NameOfSport	varchar(50)	-		Název sportu
CountOfPlayers	int	-		Počet hráčů sportu
Description	varchar(500)	-		Popis sportu
Rules	varchar(500)	-		Pravidla sportu
Collective	int	0...1		Určuje, zda se jedná o týmový sport

Tabulka Teams

Název atributu	Datový typ	Integritní omezení	Index	Popis
NameOfTeam	vardar(50)	-	primary	Primární klíč
HistoryOfTeam	varchar(500)	-		Historie týmu
Country	varchar(30)	-		Země původu týmu
IdSport	int	-		Cizí klíč, ID sportu

Tabulka Players

Název atributu	Datový typ	Integritní omezení	Index	Popis
IdPlayer	int	-	primary	Primární klíč
NameOfPlayer	varchar(50)	-		Jméno hráče
SurnameOfPlayers	vardar(50)	-		Příjmení hráče
AgeOfPlayer	int	-		Věk hráče
CountryOfPlayer	varchar(500)	-		Země původu hráče
NameOfTeam	varchar(50)			Cizí klíč, ID týmu

Tabulka Matches

Název atributu	Datový typ	Integritní omezení	Index	Popis
IdMatches	int	-	primary	Primární klíč
DateOfMatch	varchar(20)	-		Datum konání zápasu
ResultOfMatch	varchar(20)	-		Výsledek zápasu
NameOfTeamHome	varchar(50)	-		Domácí tým
NameOfTeamVisit	varchar(50)	-		Hosté tým
IdSport	int			Cizí klíč, ID sportu

Tabulka Articles

Název atributu	Datový typ	Integritní omezení	Index	Popis
IdArticle	int	-	primary	Primární klíč
HeaderOfArticle	varchar(50)	-		Nadpis článku
TextOfArticle	varchar(500)	-		Text článku
DateOfArticle	varchar(20)	-		Datum napsání
IdSport		-		Cizí klíč, ID sportu

Tabulka Discussion

Název atributu	Datový typ	Integritní omezení	Index	Popis
IdDiscussion	int	-	primary	Primární klíč
HeaderOfDiscussion	varchar(50)	-		Nadpis příspěvku
TextOfDiscussion	varchar(500)	-		Text příspěvku
DateOfDiscussion	varchar(20)	-		Datum přidání
Username	varchar(30)	-		Už. jméno
IdArticle	int			Cizí klíč, ID článku

Tabulka MatchesNo

Název atributu	Datový typ	Integritní omezení	Index	Popis
IdMatchesNo	int	-	primary	Primární klíč
DateOfMatchNo	varchar(20)	-		Datum konání
ResultNo	varchar(20)	-		Výsledek zápasu
NameOfPlayer1	varchar(30)	-		Jméno hráče
NameOfPlayer2	varchar(30)	-		Jméno protihráče
IdSport	int			Cizí klíč, ID sportu

Tabulka PlayersNoCollective

Název atributu	Datový typ	Integritní omezení	Index	Popis
NameOfPlayerNo	int	-	primary	Primární klíč
AgeOfPlayerNo	varchar(5)	-		Věk hráče
CountryOfPlayerNo	varchar(30)	-		Země původu hráče
CarierOfPlayerNo	varchar(500)	-		Kariéra hráče
IdSport	int	-		Cizí klíč, ID sportu